Synthèse des commandes SQL/DML

Cette synthèse reprend l'essentiel des commandes de la partie *Data Manipulation Language* de SQL. Je ne reprends pas les alias de colonnes ou de tables. Notons les signes de métalangages :

Signe	Explication
`[Element]`	Elément facultatif
`[Element]+`	Elément facultatif, éventuellement répété
`Element | Element`	Eléments alternatifs (l'un OU l'autre)

Requête de base

```
SELECT [DISTINCT|ALL]Champ1[,Champ]+|*
FROM Table
WHERE Condition
ORDER BY Champ [ASC|DESC][,Champ [ASC|DESC]]+|Nombre[,Nombre]+
```

Conditions

```
Champ COMPARATEUR ValeurComparaison
Champ BETWEEN Valeur AND Valeur
Champ LIKE ChaineAvecJoker
Champ IN Liste
Champ IS NULL
NOT Condition
Condition AND|OR Condition
```

Requêtes ensemblistes

Union

```
Requête1
UNION [ALL]
Requête2
[ORDER BY Critères]
```

Intersection

```
Requête1
INTERSECT
Requête2
[ORDER BY Critères]
```

Différence

```
Requête1
MINUS
Requête2
[ORDER BY Critères]
```

La norme SQL prévoit `EXCEPT` au lieu de `MINUS`.

Jointures

Jointure simple

```
SELECT ListeDeChamps
FROM Table1
INNER JOIN Table2
ON ConditionDeJointure
```

On peut évidemment ajouter une clause `WHERE` et/ou `ORDER`.

Conditions de jointure

```
ON Condition
USING (Champ[,Champ]+)
```

Je ne mentionne pas `NATURAL JOIN`, trop dangereux à utiliser.

Jointure gauche ou droite

```
SELECT ListeDeChamps
FROM Table1
LEFT|RIGHT [OUTER] JOIN Table2
ON ConditionDeJointure
```

Requêtes statistiques

Cas simple

```
SELECT FonctionStat [,FonctionStat]+
FROM Table
[WHERE Condition]
[ORDER BY Critères]
```

Fonctions

```
COUNT(*) -- Compte les lignes
COUNT(Champ) -- Compte les valeurs non nulles dans le champ
COUNT(DISTINC Champ) -- Compte les valeurs différentes
SUM(Champ) -- Somme des valeurs numériques
AVG(Champ) -- Moyenne des valeurs numériques
MAX(Champ) -- Valeur maximale
MIN(Champ) -- Valeur minimale
```

Variante avec groupement

```
SELECT Champ [,Champ]+,FonctionStat[,FonctionStat]+
FROM Table
[WHERE Condition]
GROUP BY Champ [,Champ]+
[ORDER BY Critères]
```

Filtrage des résultats

```
SELECT Champ [,Champ]+,FonctionStat[,FonctionStat]+
FROM Table
[WHERE Condition]
GROUP BY Champ [,Champ]+
HAVING ConditionPortantSurLesStatistiques
[ORDER BY Critères]
```

Requêtes imbriquées

Une requête imbriquée (un requête entourée de parenthèses) peut figurer partout où son résultat pourrait se trouver. Deux cas typiques :

- une requête produisant une cellule unique peut remplacer n'importe quelle constante
- une requête peut prendre la place d'un nom de table dans une autre requête

Requêtes imbriquées dans des conditions

```
WHERE Champ Comparateur (RequeteCellule)
WHERE (Champ,[Champ]+) Comparateur (RequeteMultiCellules)
WHERE Champ IN (RequeteMultilignes)
WHERE EXISTS (RequeteNonNulleOuNulle)
WHERE Champ Compateur ALL|ANY (RequeteMultilignes)
```

Troisième partie

SQL (DDL et DCL)

Chapitre 9

Création d'une base de données

9.1 Approche théorique

Dans une application d'une certaine ampleur, la création d'une base de données va faire l'objet d'un travail très organisé et éventuellement réparti entre plusieurs intervenants :

- analyses *conceptuelle* et *organisationnelle* menées par une équipe d'analystes : cette étape préalable est vue dans le cours d'analyse et ne sera pas rappelée ici.
- traduction du modèle organisationnel en un modèle logique : cette traduction peut être triviale dans les cas simples, mais nécessitera parfois un profond travail de réorganisation. Faute de temps, cet aspect sera principalement réservé au cours d'analyse, bien qu'on puisse incidemment en évoquer quelques aspects dans les pages qui suivent. Le modèle logique, si on a opté pour l'utilisation d'un SGBDR, sera décrit au moyen d'un langage relationnel, SQL sans doute. Afin d'éviter certaines incompatibilités, on pourra utiliser les caractéristiques propres du SGBD afin de traduire le mieux possible les contraintes du niveau conceptuel. Il existe de nombreux outils de modélisation qui effectuent cette traduction de manière automatique, en permettant souvent le choix d'un dialecte précis[1].
- le niveau physique devra se soucier de contraintes plus sordides comme :
  - les liens avec le système d'exploitation
  - l'optimisation
  - la répartition physique des données sur différentes machines
  - la création et la gestion des fichiers.

La schématisation qui vient d'être évoquée est inspirée du cycle d'abstraction de la méthode Merise. Une autre approche distingue trois niveaux qui coexistent à tout moment dans une base de données, c'est ce qu'on a appelé le modèle ANSI/Sparc.

a) le schéma externe Il s'adresse à l'utilisateur, qui peut être un simple usager de terminal ou un programmeur d'applications. Dans les deux cas, il est un consommateur de ressources informatiques qui lui sont présentées avec une certaine structure, structure qu'il ne peut en aucun cas modifier. Le schéma externe correspond à des MOD locaux implémentés. Par exemple, dans un schéma externe, on exposera des propriétés différentes d'une même entité selon le type d'utilisateur. Notons que ce schéma comporte aussi bien des données (présence ou absence de telle table, de tel attribut) que des traitements (contraintes, valeurs par défauts, mises à jour en cascade ou automatiques). Dans des cas extrêmes, le schéma externe peut se démarquer fortement du schéma conceptuel : par exemple, une base de données relationnelle présentera une vue sous forme de hiérarchie pour s'interfacer avec un programme écrit en COBOL.

[1] C'est le cas de *dbMain* et de *WinDesigne*, deux logiciels que nous utilisons au laboratoire.

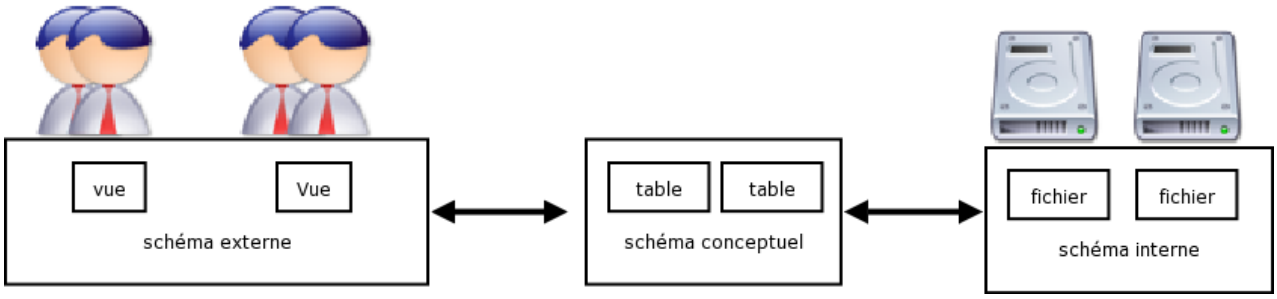


FIG. 9.1 – le modèle ANSI/Sparc

b) le schéma conceptuel Il s'agit d'une représentation de la structure de la base de données, vue indépendamment de son implémentation physique. Ce niveau est connu des gestionnaires de la base de données. Leur mission est de le créer et de le gérer de manière à procurer un schéma externe invariable (ou simplement variable par enrichissement). Toute modification du schéma conceptuel qui entraîne une remise en cause du schéma externe va perturber les utilisateurs et nécessiter la modification ou la réécriture des applications.

c) le schéma interne Il concerne très peu de personnes mais nécessite le plus de talent. Le schéma conceptuel reste une illusion, puisque seuls les fichiers physiques sont réellement mis en oeuvre par les programmes et que le niveau de performance va en dépendre. On trouvera ici de nombreuses tâches qui sont principalement du ressort de l'administrateur de la base de données.

Force est de constater que l'intervention de l'administrateur de la base de données se situe à la fois au niveau du schéma conceptuel et du schéma interne. Nous verrons que ce dernier peut céder certains de ses droits à ses adjoints. On trouvera parmi ses tâches :

- la réservation de ressources de mémoire de masse au profit des bases de données ;
- la création et la gestion des utilisateurs et des groupes (qui peut se faire en harmonie avec les groupes créés au niveau du système d'exploitation) ;
- la création des bases de données ;
- la création des tables (énumération de leurs champs, mais aussi de toutes les contraintes qui contribuent à en maintenir la cohérence) ;
- l'attribution des droits aux différents groupes pour leur permettre d'accéder aux données ;
- l'optimisation des performances : qui commence par la création d'index, le travail sur la structure physique (répartition des données dans les fichiers, emplacement physique des fichiers, modification des paramètres du système) et se termine avec l'adoption d'un nouveau SGBD (nouvelle version ou migration) ;
- la définition d'une politique de sécurité, ici encore en concertation avec les gestionnaires du réseau.

9.2 Création des bases de données

9.2.1 Réservation des ressources

La création d'une base de données avec un petit système, comme Access ou InterBase, ne pose pas trop de problème. Il suffit de donner un nom de fichier pour contenir les données et on peut se mettre au travail. Sur un serveur plus lourd, comme Oracle ou SQL Server, les

choses se compliquent par l'existence de niveaux intermédiaires de stockage. Trois méthodes sont envisageables :

a) la base de données est liée à un fichier/répertoire :

C'est l'optique d'Access (un fichier) ou de mySQL (un répertoire). Toutes les tables seront confinées dans l'espace prévus (le fichier ou le répertoire). Cela entraîne en principe la conséquence de limiter la taille de la base de données à la somme des tailles de fichiers, c'est-à-dire en définitive à la taille d'un disque dur. Avec quelques astuces (des tables liées), on pourrait dépasser cette limitation, au prix d'une détérioration des performances.

S'il n'y a pas de serveur (Access) et qu'on veut utiliser une telle base de données en réseau, on devra mettre les utilisateurs distants en contact avec le fichier ou le répertoire, ce qui va poser des problèmes de permissions, à gérer uniquement avec le gestionnaire de réseau.

b) la base de données est créé par un serveur sur une ressource virtuelle :

MS SQL-Server 6.x[2] crée des unités, qui sont en fait des réservations de place sur le disque, dans lesquels les bases de données seront créées. Fondamentalement, une base de données est créée dans deux unités : l'une pour les données principales, l'autre pour les données de journalisation, c'est-à-dire les traces de chaque modification apportée aux données. Un accroissement de la base de données pouvait se faire par un accroissement de l'unité (jusqu'à la taille d'un disque physique) ou par l'attribution d'une unité supplémentaire.

Indépendamment des unités, la base de données est enregistrée à l'intérieur d'un serveur qui est vu par tous les clients sur le réseau. La connexion avec la base de données se fait donc à travers un système de protection géré par le serveur et indépendamment du système d'exploitation.

C'est également, *grosso modo*, la solution adoptée par Oracle qui définit des « table spaces » logiques qui sont constitués par un ou plusieurs fichiers. Un utilisateur, même s'il crée des tables, ne doit donc pas préciser dans quels fichiers seront stockées ses données. Il ne peut pas, en général, choisir lui-même son « table space ».

c) la base de données est liée à plusieurs fichiers

Lors de la création de la base de données, on spécifie un ou plusieurs fichiers qui serviront à contenir les données (il est possible d'en ajouter d'autres par la suite). On retrouve cette technique dans InterBase et dans MS SQL-Server 7.0.

9.2.2 Création de la base de données

La norme SQL ne précise pas la notion de base de données et encore moins de méthode pour en créer. La création d'une base de données se fait

- soit implicitement par l'installation d'un serveur sur une machine (DB2). Il n'y a en conséquence qu'une base de données par serveur.
- soit par l'invocation d'un programme spécifique (Ingres ou Oracle)

[2]Cette technique a été abandonnée à partir de la version 7.0. Essentiellement, il s'agissait pour les ingénieurs de SQL Server de palier les carences du système d'exploitation en permettant de réserver des ressources plus grandes qu'un disque ou plus sûres. Le programme fournissait donc des volumes répartis sur plusieurs disques physiques avec ou sans redondance (RAID logiciel). Les fonctionnalités offertes par le système d'exploitation à partir de Windows NT 4 ont poussé les développeurs de SQL Server à abandonner l'implémentation des volumes.

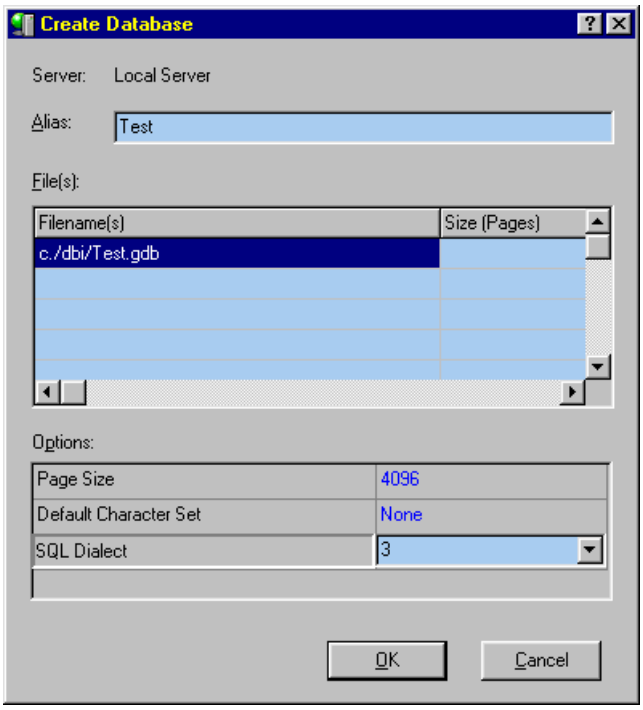


FIG. 9.2 – Création d'une base de données avec InterBase

– soit encore par l'instruction `CREATE DATABASE`, dont la syntaxe varie dans le détail d'un système à l'autre.

Voici à titre informatif, la méthode employée par InterBase pour créer une nouvelle base de données. On peut y créer une base de données soit par l'interface graphique (voir figure 9.2), soit par un script ou encore en tapant des commandes.

Voici un exemple pratique - et qui fonctionne ! - :

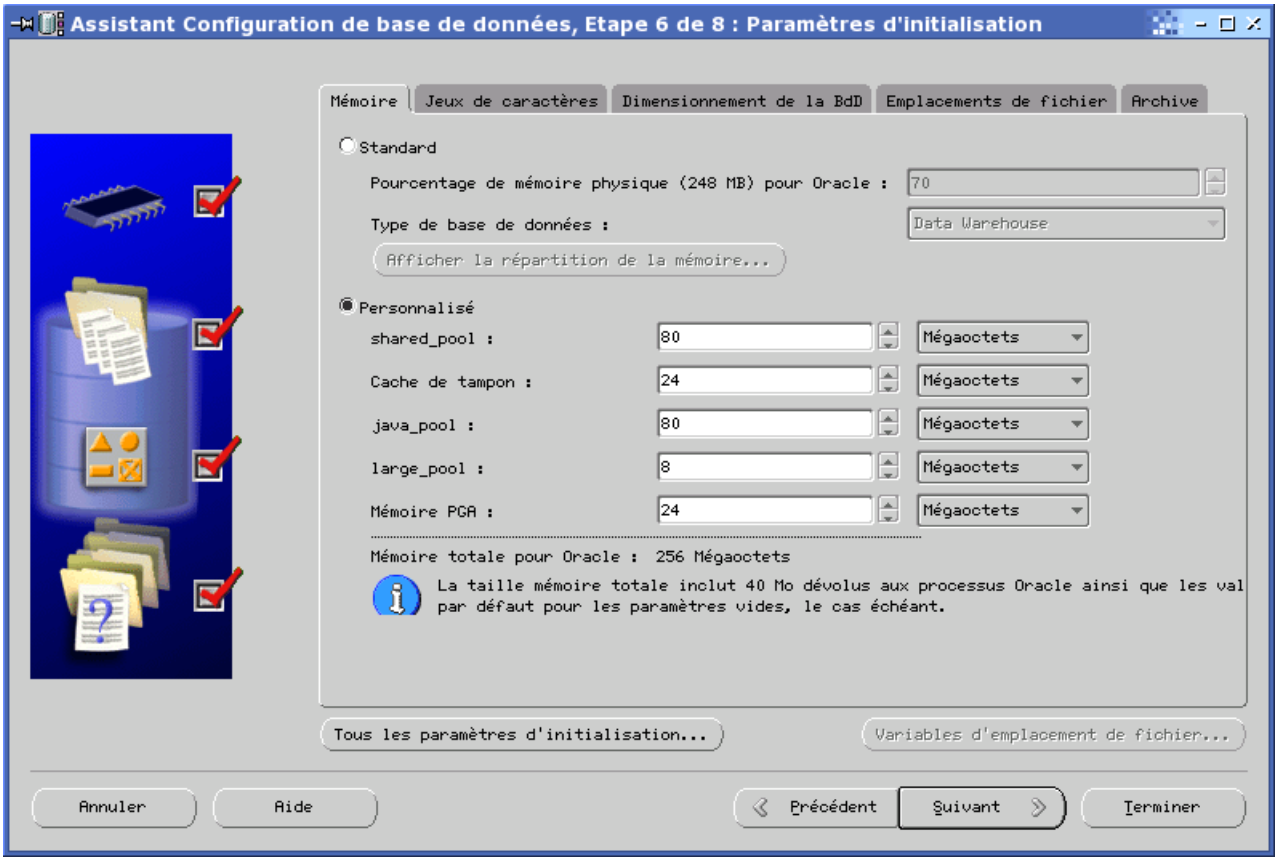
```
/* Nouvelle base de données de test */
CREATE DATABASE "c:/dbi/Test.gdb" PAGE_SIZE 1024
USER "JACQUES" PASSWORD "1234"
;
```

La base de données sera créée ici dans un seul fichier dont on donne le chemin exact. On peut préciser la taille (LENGTH), la taille des pages (PAGE_SIZE) et l'ensemble de caractères par défaut (utile pour les langues à caractères spéciaux). L'utilisateur mentionné sera le propriétaire de la base de données.

SQL-Server propose une méthode à peu près semblable. Oracle utilise une méthode lourde de création à l'aide d'un utilitaire spécifique, dont l'exécution finale prend une bonne vingtaine de minutes (il s'agit du programme `dbca`). En pratique, il parfaitement possible de gérer un système Oracle avec une seule base, puisque l'existence des schémas, appartenant à des utilisateurs différents correspond en fait à des mini-bases de données internes et indépendantes. Il est cependant très facile d'établir des liens entre plusieurs schémas.

Dans des environnements professionnels, si on ne dispose que d'un seul serveur, on peut envisager d'avoir deux bases de données : une base de données de production, utilisée en temps réel par les employés de l'entreprise, et une base de données expérimentale qui permet de tester les futures modifications. Cette double base de données (ou ce double ensemble de bases de données) se retrouve dans d'autres systèmes.

L'illustration suivante montre l'un des huit écrans de l'assistant de création de base de données d'Oracle 9i.



9.2.3 Les schémas d'Oracle

Si l'utilisation de plusieurs bases sur un même serveur n'est pas possible ou pas souhaitée, on peut utiliser les *schémas*. Oracle utilise cette technique. Lorsqu'un utilisateur se voit donner le droit de créer des tables, on doit également lui attribuer un *tablespace* et un quota. La commande suivante crée un utilisateur et lui donne ces droits :

```
GRANT CONNECT,RESOURCE,UNLIMITED TABLESPACE
TO GARAGE IDENTIFIED BY CODD;
```

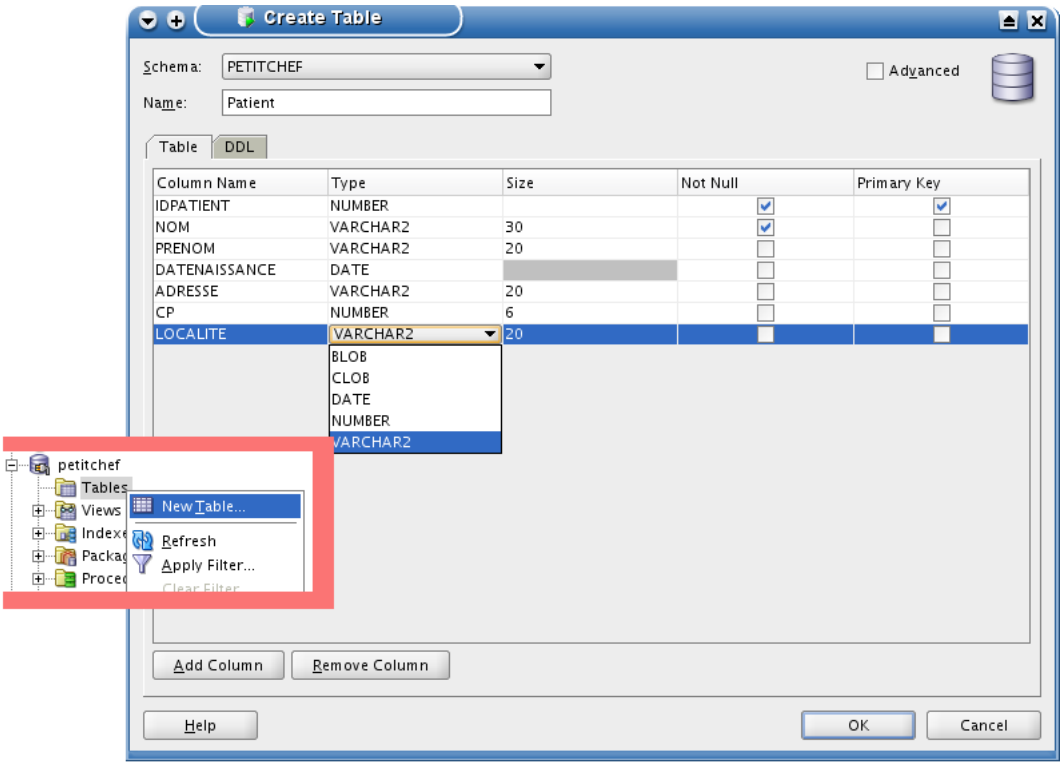
Lorsque l'utilisateur lancera la commande `CREATE TABLE` vue dans la prochaine section, un nouveau schéma va être créé pour lui, qui portera son nom. Il pourra créer et manipuler les tables de son schéma et, éventuellement, donner à d'autres utilisateurs le droit de les consulter ou de les modifier. L'utilisateur devient ainsi une sorte de petit administrateur de son schéma. Les droits que nous venons d'examiner ne lui permettent cependant pas de s'offrir des adjoints pour créer ou modifier ses tables.

9.3 Création des tables

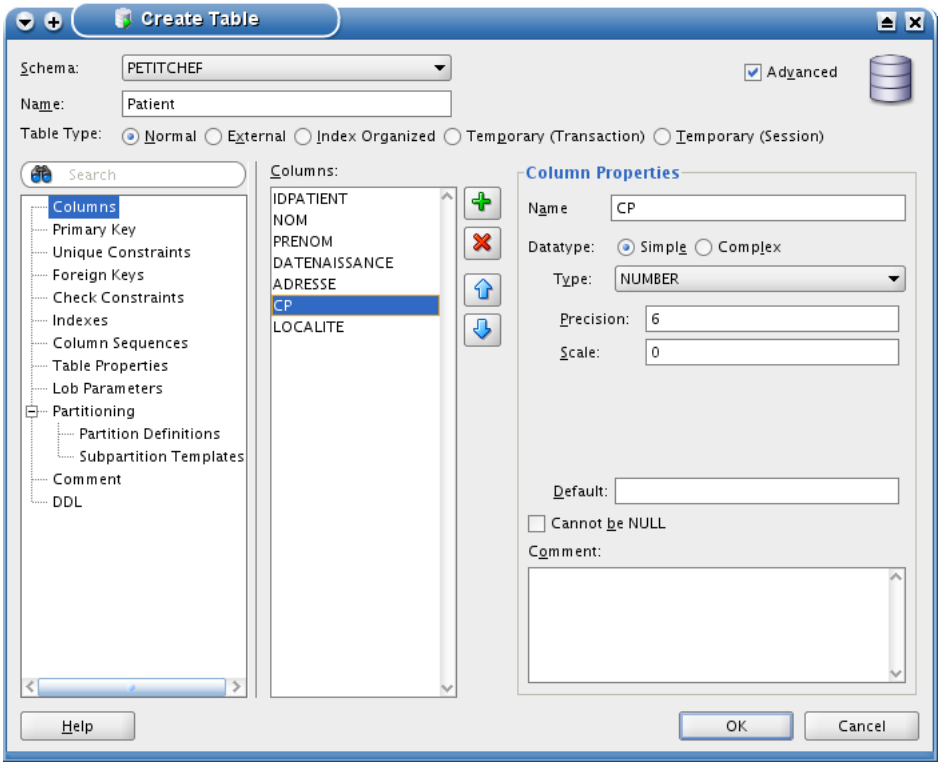
Notre approche se limitera à la définition des champs. Nous reviendrons sur les contraintes, au sens large dans le chapitre suivant. Nous laisserons même tomber provisoirement les clés primaires, qui sont pourtant obligatoires dans une perspective relationnelle.

9.3.1 Utilisation d'une interface graphique

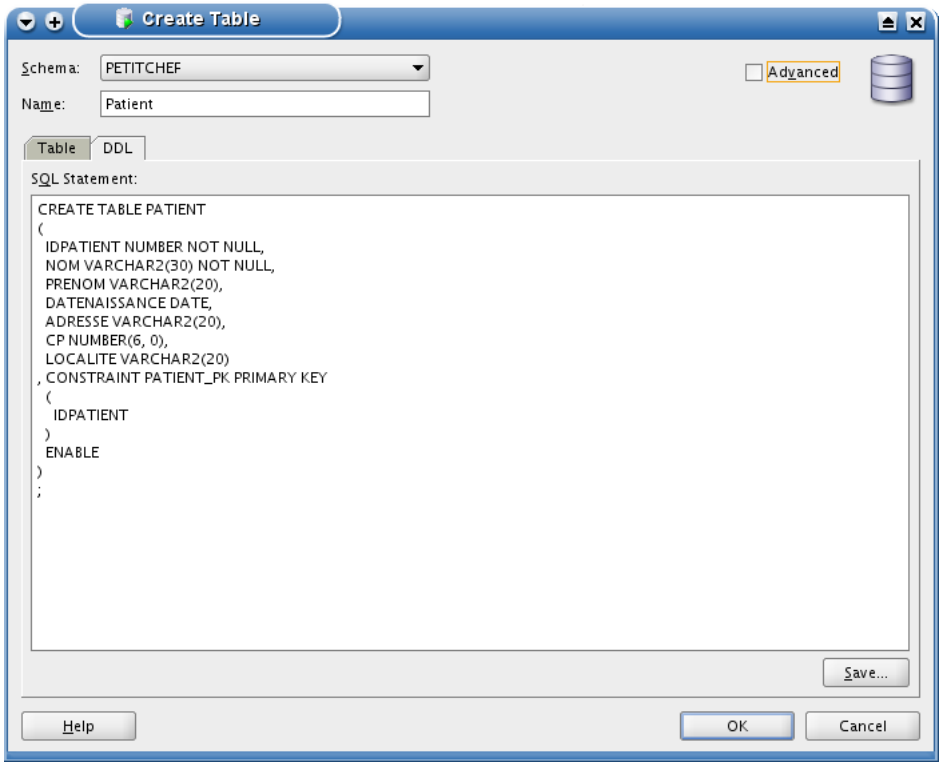
SQL Developper permet au propriétaire d'un schéma de créer ses tables à l'aide d'une interface graphique. Cette technique est évidemment plus facile puisqu'il suffit de choisir les bonnes options. Il faut néanmoins se rappeler que la séquence de touches et de clics d'un tel travail ne peut pas être répétée (par exemple sur la machine de production après une phase de test). Il convient donc de vérifier si le système permet d'exporter un script réutilisable. C'est évidemment le cas de notre utilitaire favori.



Le mode « advanced » offre encore plus de possibilités.



A tout moment, on peut voir le code SQL correspondant (celui qui sera de toute manière envoyé au serveur).



9.3.2 Syntaxe de la création d'une table

La commande de création d'une table se compose des éléments suivants :
- un en-tête `CREATE TABLE` suivi du nom de la table (éviter les caractères exotiques, surtout si on envisage de partager les tables entre plusieurs systèmes) ;
- une liste de définitions de champs séparées par des virgules, liste mise entre parenthèse (chaque champ possède un type, que nous étudierons dans le paragraphe suivant) ;
- des contraintes globales placées après la liste des champs (que nous verrons plus tard).

```
CREATE TABLE NomTable
( champ1 type1,
  champ2 type2,
...
  champN typeN,
  contrainte1,
...
  contrainteN
);
```

Exemple :

```
CREATE TABLE Etudiants
( Id_Etudiant INTEGER NOT NULL,
  NomEtudiant VARCHAR(30) NOT NULL,
  PrenomEtudiant VARCHAR(20) NOT NULL,
  Adresse VARCHAR(50) ,
  DateNaissance DATE,
```

```
    CONSTRAINT pkEtudiant PRIMARY KEY (Id_Etudiant)
);
```

9.3.3 Domaines ou types de données

Les bases de données relationnelles sont, comme les langages de programmation, dotées de types[3]. Ces types présentent deux avantages :

- permettre de prévoir l'emplacement occupé par chaque donnée (sauf pour certains types à taille variable, pour lesquelles la gestion de l'espace est plus complexe) ;
- autoriser un contrôle de validité des données lors de l'encodage (types numérique ou date).

SQL présente des types standard qu'on ne retrouve pratiquement jamais tels quels dans les implémentations. On veillera à consulter la documentation du SGBDR qu'on utilise. Les tableaux suivants essaient de montrer les correspondances entre SQL2 et les implémentations InterBase, SQL Server et Access de Microsoft (la taille en octet est toujours dépendante de l'implémentation, on donne ici celle de SQL Server)[4] . Oracle utilise un système original de types, mais accepte toutes les spécifications du standard SQL2, qu'il traduit dans son système de manière transparente. Les types d'Oracle ne figureront donc pas dans ces tableaux.

a) Nombres entiers :

SQL ISO	InterBase	SQL Server	Access	Description

`INTEGER`
`INT`
	`INTEGER`	`INTEGER`
		`INT`
			`INTEGER`
			`INT`
			`INT4`
				Entier 4 octets

`SMALLINT`
`INT`
	`SMALLINT`	`SMALLINT`	`SMALLINT`
			`SHORT`
			`INTEGER2`
				Entier 2 octets

		`TINYINT`	`BYTE` `INTEGER1`	Entier 1 octet

		`BIT`	`BIT`
			`BOOLEAN`
			`LOGICAL`
				Valeur binaire
				1 octet peut
				contenir 8 champs

Oracle ne fait aucune différence entre `INTEGER` et `SMALLINT`. Il les représente en interne par `NUMBER(38)` qui est son plus grand format d'entiers. Il permet d'autres formats plus petits en spécifiant le nombre de chiffres significatifs. La taille occupée en mémoire est légèrement supérieure à la moitié du nombre de chiffres. Ex. `NUMBER(1)`, `NUMBER(6)`, `NUMBER(10)`, `NUMBER(38)`...

ORACLE	Entiers	`NUMBER(N)`

[3] Une notable exception est SQLite, le SGBD léger fourni avec PHP 5.

[4] Si, comme l'an dernier, nous sommes amenés à utiliser également le système mySQL, il aurait fallu également parler de la syntaxe et des types propres à ce système. Cela aurait rendu les tableaux illisibles. Si besoin est, je donnerai des informations sur mySQL en annexe.

b) Décimal condensé

SQL ISO	InterBase	SQL Server	Access	Description
DECIMAL(p,s) DEC(p,s) NUMERIC(p,s)				Décimal exact 2 à 17 octets.
		MONEY	CURRENCY MONEY	Nombre à 4 décimales fixes (8 octets)
		SMALLMONEY		Idem à 4 octets.

Le paramètre *p* précise le nombre total de chiffres, *s* précise le nombre de chiffres après le point décimal. Les paramètres sont facultatifs, mais les valeurs par défaut pouvant varier d'un système à l'autre, cette pratique doit se déconseiller. J'y ai ajouté les formats MONEY et SMALLMONEY de Microsoft, qui sont construits sur le même principe (sans paramètres), mais affichent automatiquement le symbole monétaire.

Oracle se conforme au standard, mais utilise en interne son synonyme NUMBER. Ex. NUMBER(15,2).

ORACLE	Décimaux	NUMBER(p,s)

c) Nombres pseudo réels

SQL ISO	InterBase	SQL Server	Oracle	Description
FLOAT(n)		FLOAT(n)	FLOAT(n)	Pseudo-réels avec choix de la définition
DOUBLE PRECISION				Pseudo-réel sur 8 octets
		FLOAT(8) -> FLOAT(15)	FLOAT	Pseudo-réel sur 8 octets
REAL	FLOAT	REAL FLOAT(1) -> FLOAT(7)	REAL	Pseudo réel sur 4 octets

On remarquera que bien qu'il autorise une précision apparemment confortable, SQL-Server réduit en fait tout à des nombres comportant 7 ou 15 chiffres significatifs. Access n'a que des pseudo-réels sur 4 octets, qu'il nomme SINGLE, FLOAT4 ou REAL. À noter qu'à la différence de SQL-Server, Oracle note sa précision en bits (REAL est équivalent à FLOAT(63)).

ORACLE	Réels	FLOAT(b)

d) Texte

Ici, Access fait bande à part en n'utilisant qu'un type TEXT, dont on peut préciser le nombre de caractères. Tous les autres systèmes se conforment à la norme SQL2 en proposant quatre familles de données définies par deux critères :

1. les données sont de taille fixe (par défaut) ou de taille variable. On précise dans les deux cas la taille maximale admise. Si la taille est fixe, la zone est remplie d'espaces. Sinon, l'espace non utilisé n'est pas consommé. Cette solution, qui offre de moins bonnes performances, est à privilégier si on dispose de machines puissantes.

2. les données sont représentées à l'aide d'un octet par caractère, solution traditionnelle, ou par plusieurs, ce qui autorise les alphabets « nationaux »[5].

Norme ISO	Abréviation	ORACLE
CHARACTER(*n*)	CHAR(*n*)	CHAR(*n*)
NATIONAL CHARACTER(*n*)	NCHAR(*n*)	NCHAR(*n*)
CHARACTER VARYING(*n*)	VARCHAR(*n*)	VARCHAR2(*n*)
NATIONAL CHARACTER VARYING(*n*))	NVARCH(*n*)	NVARCHAR2(*n*)

```
CREATE TABLE Octobre.Test(
        /* Champs de taille fixe */
        Nom      CHARACTER(15),
        Prenom   NATIONAL CHARACTER(15),
        /* Champs de taille variable */
        Adresse  CHARACTER VARYING(15),
        Ville    NATIONAL CHARACTER VARYING(15));
```

Il faudrait encore parler du codage des caractères, mais il dépend des paramètres de création de la base de données. Unicode tend à se généraliser. On précise alors le type de code utilisé (pour le français, c'est généralement ISO8859_1 ou ISO8859_15 qui dispose en outre du symbole de l'euro). L'utilisation d'Unicode au sein d'Oracle est tellement complexe, qu'un document spécial[6] y est consacré. Le problème concerne surtout les langues asiatiques.

e) Heures et dates

SQL 2	InterBase	Microsoft	Description
DATE	DATE		Uniquement la date
TIME			Uniquement l'heure
TIMESTAMP		DATETIME	Date et heure
		SMALL DATETIME	Date et heures 4 octets

On constate que Microsoft ne sait pas séparer la date et l'heure (Access reconnaît en plus TIMESTAMP). Oracle ne permet pas d'exprimer l'heure sans la date. L'emploi des dates se révèle particulièrement perturbant quant on passe d'un système à l'autre. On constate par exemple une relative incompatibilité entre InterBase et SQL Server : le premier utilise un type DATE et le second un type DATETIME qui contient également l'heure. Lors des comparaisons, il faudra faire attention à la présence de cette heure qui peut empêcher des égalités de se produire.

ORACLE		
	Date	DATE
	Date et heure	TIMESTAMP

[5] Je ne parlerai pas ici d'Unicode, qui comme son nom ne l'indique pas regroupe plusieurs standards de représentation des caractères internationaux. Dans nos pays occidentaux, on utilise souvent UTF-8 dans lequel la taille d'un caractère varie entre 1 et 2 octets.

[6] Voir :*http ://www.oracle.com/technology/tech/globalization/pdf/TWP_AppDev_Unicode_10gR2.pdf*

f) Valeurs binaires

SQL 2	InterBase	SQL Server	Access	Description
BIT(n)		BINARY(n)		n octets
BIT VARYING(n)	BLOB	VARBINARY(n)	BINARY VARBINARY	un maximum de n octets

Ces types permettent de stocker des objets divers, des images. Access permet par exemple de stocker des liens OLE. Notons qu'Oracle possède un grand nombre de variantes pour ces types binaires :

LONG, RAW, LONG RAW, BLOB, CLOB, NLOB, BTILE, UROWID.

g) Le type INTERVAL

Le type INTERVAL (proposé dans SQL 2) correspond à des durées. Exemples : YEAR TO MONTH ou DAY TO SECOND. Il est implémenté par Oracle. On peut préciser un nombre de chiffres pour l'année ou le jour (YEAR(4) TO MONTH). Les intervalles peuvent être négatifs. Les valeurs des composants de droite ne peuvent jamais atteindre le maximum : 11 pour les mois, 30 pour les jours, 23 pour les heures...

À titre de synthèse, voici un exemple qui crée une table reprenant les différents types de mesure du temps dans Oracle et quelques encodages. Rappelons que les réglages nationaux des clients peuvent interférer sur la manière dont les formats sont lus et s'affichent. Je donne plus bas un moyen de contourner le problème :

```
CREATE TABLE TestDate(
        D DATE,
        TS TIMESTAMP,
        YM INTERVAL YEAR to MONTH,
        YM4 INTERVAL YEAR(4) to MONTH,
        DS INTERVAL DAY to SECOND);

INSERT INTO TestDate(D,TS)
VALUES('21 avril 2003','21 avril 2003');
INSERT INTO TestDate(D,TS)
VALUES('22 avril 2003','22/04/2003 10:11:12');
INSERT INTO TestDate(D,TS)
VALUES('23 avril 2003','23/04/03');

SELECT D,TS FROM TestDate;
D        TS
-------- ------------------------------------
21/04/03 21/04/20 03:00:00,000000
22/04/03 22/04/03 10:11:12,000000
23/04/03 23/04/03 00:00:00,000000
```

Ce premier exemple montre que si la date est bien interprétée dans le champ de type DATE, il est indispensable de préciser l'heure avec un champ de type TIMESTAMP. On voit comment les deux derniers chiffres de l'année sont confondus avec les heures dans le deuxième encodage. Si on ne veut pas taper l'heure, il faut mettre l'année sur deux chiffres. Il est possible d'utiliser une notation indépendante des réglages nationaux en utilisant les mots DATE et TIMESTAMP :

```
INSERT INTO TestDate(D,TS)
VALUES(DATE '2004-1-31',TIMESTAMP '2004-1-31 14:12:15');
```

Voyons quelques intervalles (ils peuvent être négatifs) :

```
INSERT INTO TestDate(YM) VALUES('04-01');
INSERT INTO TestDate(YM) VALUES('99-11');
INSERT INTO TestDate(YM) VALUES('-99-11');
INSERT INTO TestDate(YM4) VALUES('9999-11');
INSERT INTO TestDate(DS) VALUES('-99 23:59:59');
```

9.4 Fonctions avancées de gestion des tables

Les fonctionnalités décrites dans cette section ont été vérifiée sur Oracle. On peut les retrouver dans d'autres SGBD, avec parfois certaines variantes.

9.4.1 Création d'un schéma

Oracle utilise la notion intermédiaire de *schéma*. Les tables ne sont donc pas rangées dans la base de données, mais dans des schémas. Un schéma porte toujours le nom d'un utilisateur. Il n'existe pas d'instruction pour créer un schéma : lors de la création d'une première table par un utilisateur, un schéma se crée automatiquement. Il existe deux types de créateurs de tables, définis par deux droits différents :

- `GRANT CREATE TABLE TO Octobre` autorise l'utilisateur *Octobre* à créer des tables dans son propre schéma (et implicitement à créer son schéma)
- `GRANT CREATE ANY TABLE TO Octobre` l'autorise en outre à créer une table dans n'importe quel schéma.

Le créateur d'un schéma possède par nécessité tous les droits sur ce schéma.

9.4.2 Tables temporaires

La plupart des SGBD modernes connaissent la notion de tables temporaires. Ces tables ont des caractéristiques originales :

- leurs données ne sont visibles que pour leur créateur
- elles utilisent un espace de stockage différent des autres tables
- les données ne persistent que pendant la session de l'utilisateur

Leur utilité réside dans la création de données qui permettent de simplifier certains traitements en mémorisant des résultats intermédiaires, soit parce qu'il est difficile de s'en passer, soit parce qu'ils sont réutilisés plusieurs fois. Leur caractère temporaire facilite la tâche de l'utilisateur : il est certain de ne pas laisser traîner des données périmées, il ne gaspille pas d'espace et il ne doit pas se charger de les effacer.

Oracle précise au moment de la création si les données doivent disparaître au moment de la fin de la session ou directement lors de la fin de la transaction. Une fois créée, la table obéit aux mêmes règles que les tables normales : on peut autoriser les autres utilisateurs à s'en servir. Il faut, dans ce cas, donner des droits complets, puisqu'un utilisateur ne pourra jamais voir les données insérées par d'autres.

```
CREATE GLOBAL TEMPORARY TABLE Temporaire
(
Num int,
```

```
Nom varchar2(30)
) ON COMMIT PRESERVE ROWS ;
```

Si on veut que les données disparaissent dès la fin de la transaction, on supprime la fin de la définition (ou on la remplace explicitement par son contraire) :

```
) ON COMMIT DELETE ROWS;
```

9.4.3 Copier une table

Nous avons vu dans le chapitre précédent comment peupler une table avec des données issues d'une autre table. Pour ne pas devoir créer la structure, on peut utiliser une autre technique qui crée à la fois la table et les données.

```
CREATE [GLOBAL TEMPORARY] TABLE NomTable
AS SELECT Champ1, Champ2 ... FROM TableExistante.
```

Dans la séquence suivante, je vais créer une table reprenant les sections de l'école :

```
-- Création
CREATE TABLE Sections
AS SELECT Section FROM Etudiants;
```

Attention néanmoins :

1. la nouvelle table ne possède pas les contraintes de la table originale, il faut donc recréer la clé primaire.
2. dans le cas d'une table temporaire, les données ne sont pas toujours conservées puisque la commande `CREATE TABLE` provoque automatiquement un `commit`.

9.4.4 Modifier la structure d'une table

Ce n'est pas une bonne pratique de créer un table par essais et erreurs en ajoutant, modifiant et supprimant les champs au fur et à mesure des nécessités. Il arrive parfois, cependant, qu'au cours de l'histoire d'un système, on ait besoin de modifier les tables pour tenir compte de l'évolution du réel. En terme de performances, il est souvent conseillé de recréer une nouvelle table pour éviter une dispersion des données.

Renommer une table

En renommant une table, on risque évidemment de modifier le schéma externe et de perturber des applications existante.

```
RENAME AncienNom TO NouveauNom;
```

Ajouter un ou plusieurs champs

Dans chaque ligne, le champ contiendra la valeur par défaut (normalement Null[7]). On peut, au choix ajouter un ou plusieurs champs :

```
ALTER TABLE CopieVente
ADD Champ1 INT;
ALTER TABLE CopieVente ADD
(Champ2 int,
 Champ3 int);
```

Supprimer un champ

La suppression du champ entraîne la perte des données contenues dans ce champ pour toutes les lignes de la table. La syntaxe en est simple :

```
ALTER TABLE CopieVentes
DROP COLUMN Champ1;
ALTER TABLE COPIEVENTES DROP
(Champ2,Champ3);
```

Modifier un champ

Cette modification ne peut se faire qu'à condition que la nouvelle version de la colonne reste en mesure de contenir les données existantes. Dans notre exemple, nous partons de `Number(4,0)`. Il est possible d'élargir à 6. `Number(2,0)` provoquera une erreur.

```
ALTER TABLE CopieVentes
MODIFY Montant NUMBER(6,0);
```

Renommer un champ

Comme pour le nom de la table, il convient ici d'envisager les conséquences d'une telle action sur les applications :

```
ALTER TABLE CopieVentes
RENAME COLUMN Montant TO Amount;
```

Exemple réel

Dans l'exemple suivant, nous allons « normaliser » la table des étudiants, en travaillant toutefois sur une copie. Nous réalisons donc cette copie.

```
-- Création d'une copie pour réaliser l'exercice
--    structure
CREATE TABLE CopieEtudiants AS
SELECT * FROM Etudiants;
--  Ajout de la clé primaire
ALTER TABLE CopieEtudiants
CONSTRAINT pkCopieEtudiants PRIMARY KEY(N);
```

[7]Notons à ce sujet qu'un champ ajouté ne peut jamais avoir la contrainte NOT NULL, ce qui se comprend facilement.

Nous créons ensuite une table reprenant toutes les sections présentes dans la table principale, en évitant les doublons.

```
-- Créer une table avec les sections
--   structure
CREATE TABLE Section AS
SELECT DISTINCT Section FROM CopieEtudiants;
```

Au passage, nous décidons de renommer cette table (il y a plusieurs sections), puis de lui ajouter un champ qui servira de clé primaire.

```
-- Le nom de la table sera plus joli au pluriel
RENAME Section TO Sections;
-- Ajout du champ destiné à devenir clé primaire
ALTER TABLE Sections
ADD idSection CHAR(5);
```

Pour générer la clé primaire, nous prenons les trois premiers caractères du nom de la section et le dernier (qui indique le plus souvent l'année). Il n'y a pas de doublons (il faut quand même vérifier).

```
UPDATE Sections
SET idSection= upper(substr(Section,1,3)||substr(section,-1));
COMMIT;
```

Faisons de ce champ une clé primaire (c'est le moment où on se mord les doigts s'il y avait des doublons).

```
ALTER TABLE Sections
ADD CONSTRAINT pk_sections PRIMARY KEY(idSection);
```

Il reste à ajouter un champ à la table des étudiants et à le remplir avec des données convenables. Je pourrais utiliser les données de la table comme pour la génération de la clé primaire de *Sections*, mais je vais jouer le jeu en allant chercher la clé au moyen du nom de la section (je pourrais par exemple avoir créé les clés manuellement ou sous forme numérique).

```
ALTER TABLE CopieEtudiants
ADD idSection CHAR(5);

UPDATE CopieEtudiants
SET idSection = (SELECT idSection FROM Sections
WHERE Section=CopieEtudiants.Section);
COMMIT;

-- La colonne Section est désormais inutile, on la supprime.
ALTER TABLE CopieEtudiants
DROP COLUMN Section;

-- Vérification de la cohérence des données
SELECT * FROM CopieEtudiants
INNER JOIN Sections USING(idSection);
```

9.4.5 Situations physique et logique des tables

Oracle situe les tables dans des espaces de stockage (*table spaces*) qui sont définis par l'administrateur. Par défaut, un utilisateur nouvellement créé ne dispose d'aucune permission sur ces espaces ni d'aucun quota de création. Pour qu'il puisse effectivement créer ses tables, on doit lui donner des droits supplémentaires.

```
ALTER USER "JEAN"
DEFAULT TABLE SPACE "USERS"
QUOTA 100 K ON "USERS";
```

La commande précédente donne le droit à Jean de se connecter sur l'espace *Users* et d'y consommer 100Ko. On peut évidemment définir ces paramètres lors de la création de l'utilisateur.

Indépendamment de la situation physique d'une table, il faut pouvoir la nommer dans une requête. Par défaut, tout utilisateur manipule les tables de son propre schéma. S'il veut manipuler les tables d'un autre utilisateur, à condition que celui-ci lui en ait donné le droit, il doit préfixer le nom de la table par le nom du schéma du propriétaire. Pour des tables fréquemment utilisées, on pourra définir des synonymes. Par exemple, Jean pourra choisir de créer le synonyme suivant pour manipuler la table des étudiants :

```
CREATE SYNONYM etu FOR Octobre.Etudiants
```

9.4.6 S'y retrouver dans la base

L'explorateur de SQL-Developper nous permet de naviguer dans la base de données (notamment à travers la liste « Other users »). Quand on se trouve en ligne de commandes, c'est parfois plus difficile. Oracle dispose d'un grand nombre de tables et de vues qui offrent des listes des tables, des schémas, des champs, des utilisateurs.

Liste des utilisateurs	`SELECT * FROM Dba_Users;`

Liste des schémas	Je n'ai pas trouvé la commande, mais il est possible de l'obtenir en listant tous les propriétaires de tables. C'est vrai que ça ne donne pas les schémas vides.
	`SELECT DISTINCT Owner`
	`FROM All_Tables;`

Données de l'utilisateur courant	
tables et type	`SELECT * FROM User_Catalog;`
tables et détails	`SELECT * FROM User_Tables;`
description des tables	`SELECT * FROM User_Tab_Comments;`
liste des vues	`SELECT * FROM User_Views;`
Données pour tous les utilisateurs	
tables et type	`SELECT * FROM All_Catalog;`
tables et détails	`SELECT * FROM All_Tables;`
description des tables	`SELECT * FROM All_Views;`
liste des vues	`SELECT * FROM All_Tab_Comments;`
Données pour un utilisateur précis	
`-- Ajouter` `WHERE owner='NOM_UTILISATEUR' -- en majuscules`	