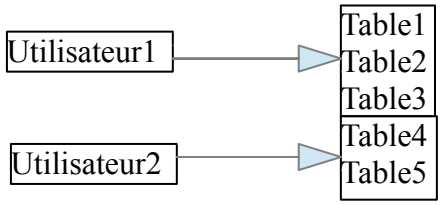
SQL – ORACLE

Oracle en Pratique

Notion de Schéma sous Oracle

C'est un ensemble comprenant des structures de données et des données, il appartient à un utilisateur et porte le nom de ce dernier.

Chaque utilisateur possède ainsi son propre schéma.



L'utilisateur 1 est propriétaires des tables (table1, table2, table3) et l'utilisateur 2 est propriétaire des tables (Table4 et Table5)

Si l'utilisateur 1 est connecté, il peut réaliser les requêtes suivantes :
SELECT * FROM Table1 ;
SELECT * FROM Table2 ;
SELECT * FROM Table3 ;

la requête suivante échouera en provoquant une levée d'exception sous Oracle :
SELECT * FROM Table4 ;

Si l'utilisateur à des droits sur la table4, il devra utiliser la requête en qualifiant le nom de la table par le nom du propriétaire (schéma).

SELECT * FROM Utilisateur2.Table4 ;

De même pour l'utilisateur 2 si il a des droits d'accès à la Table2, il devra réaliser la requête de la manière suivante :

SELECT * FROM Utilisateur1.Table2 ;

Utiliser SQLPLUS

SQL*Plus est un outil de réalisation de requêtes et de commande interactive (BATCHS) qui est installé avec toutes les installations d'Oracle.
C'est un outil en ligne de commandes, qui se décline également en un outil GUI et une interface Web nommée iSQL*Plus.
Il existe également un "Instant Client SQL*Plus" qui est un client Stand-Alone en ligne de commande disponible sur les plates-formes qui supportent l'Instant Client OCI.

SQL*Plus a son propre environnements de commandes et permet un accès aux base de données Oracle.
Il permet de saisir et exécuter des commandes SQL, PL/SQL, SQL*Plus et des commandes systèmes et ainsi :

- Permet de Formater, de calculer, stocker, et afficher les résultats des requêtes.
- D'examiner les tables et les définitions des objets.
- Développer et exécuter des scripts batch.

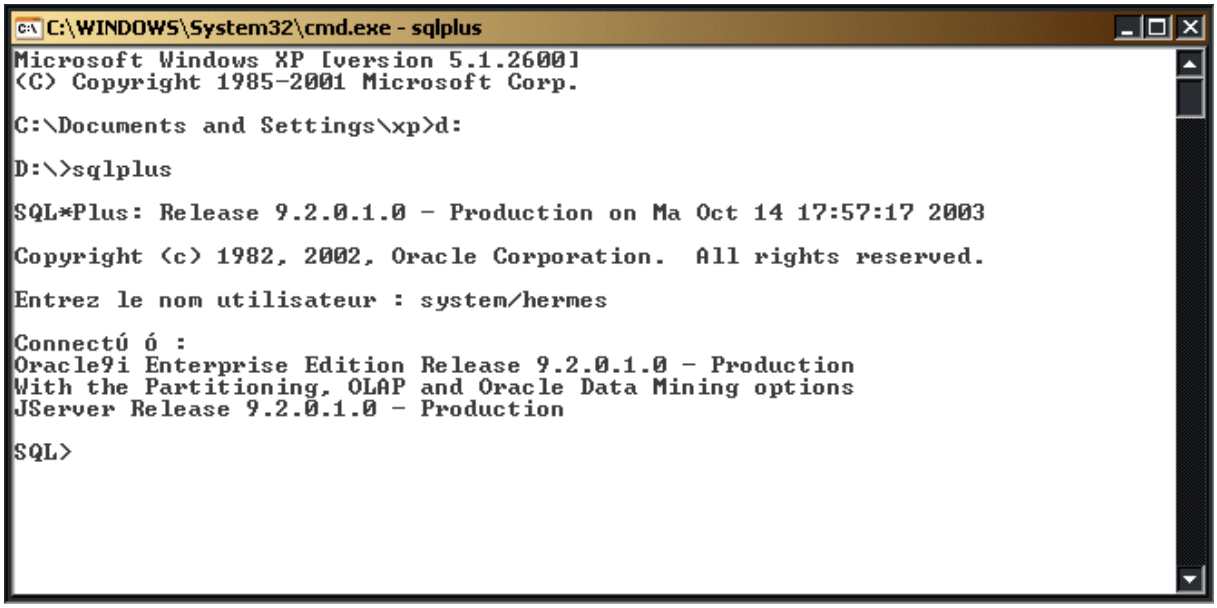
- Permettre l'administration des bases de données.

Vous pouvez utiliser SQL*Plus pour générer des rapports interactives, générer des rapports par processus batch,
et sortir le résultat dans des fichiers textes ou vers l'écran ou vers des pages HTML.

SQLPLUS, est un outil de requête en ligne de commande, il est présent depuis les premières versions d'ORACLE, toujours présent dans la version 11g d'Oracle, on l'utilisera plutôt comme un outil permettant de réaliser des requêtes par BATCHS(Tâches automatisées) ou pour dépanner en cas de problèmes.
Cet outil fonctionne selon la philosophie des outils VI ou EMACS.

Se connecter à SQL*Plus
S'asurer que le fichier tnsnames.ora défini bien le net service Name (NSN)
S'assurer que la variable ORACLE_HOME est correctement définie
Connaitre le nom et mot de passe du propriétaire du schéma de la BDD
SQLPLUS [connexion] [@fichier_script [argument [,…]]]

SQL*Plus schéma/password@NomBase
ou
SQLPlus¬
CONNECT schéma/password@NomBase



Une fois une session SQL*PLUS débutée, l'utilisateur peut travailler en interactif ou non. Dans le premier cas il saisira ses commandes sous le prompt SQL et devra les terminer par le caractère « ; » pour lancer l'interprétation.
Dans le second cas, il construit ses scripts (avec l'extension « .sql ») et les lance sous le prompt SQL en les faisant précéder de start ou @. Une session SQL*PLUS se termine par la commande exit. La transaction en cours est alors validée.
Une requête peut s'écrire sur plusieurs lignes. A chaque retour chariot l'interpréteur incrémente le numéro de ligne jusqu'au « ; » final qui marque la fin de la saisie.

Les commandes dans SQLPLUS

- **COL ADRESSE FORMAT A20**, formater l'affichage d'une colonne ADRESSE sur 20 caractères

- **COL PRIXUNIT FORMAT 99.99**, formater l'affichage d'une colonne PRIXUNIT

- **CLEAR COL**, ré-initialiser la taille des colonnes par défaut

- **SET LINESIZE 100**, reformater la taille de la ligne à 100 caractères

- **SET PAUSE ON**, afficher un résultat page par page

- **SHOW USER**, visualiser le user sous lequel on est connecté

- **CONNECT** , se connecter à l'instance User/MotPass@adresseServeur , permet de changer de session utilisateur

- **CLEAR SCREEN**, ré-initialiser l'écran

- **SET SQLPROMPT TEST>** , afficher le prompt SQL en : TEST>

- DESC Nom_Table, afficher la structure d'une table ou d'une vue

- **@ nom_ficher**, permet d'exécuter le contenu d'un fichier sql

- /, ré-active la dernière commande

- **SET ECHO ON/OFF**, affiche ou non le texte de la requête ou de la commande à exécuter

- **SAVE nom_ficher [append|create|replace]**, permet de sauvegarder le contenu du buffer courant dans un fichier « .sql ».

- **TIMING ON|OFF**, provoque l'affichage d'informations sur le temps écoulé, le nombre d'E/S après chaque requête

- **TI ON|OFF**, provoque l'affichage de l'heure avec l'invite

- **TERM [ON|OFF]**, supprime tout l'affichage sur le terminal lors de l'exécution d'un fichier

- **VER [ON|OFF]**, provoque l'affichage des lignes de commandes avant et après chaque substitution de paramètre.

- **SQL }**, spécifie le caractère « } » comme étant le caractère de continuation d'une commande SQL*Plus.

- **SUFFIX txt**, spécifie l'extension par défaut des fichiers de commande SQL*Plus

- **GET fichier** Charge dans le buffer le contenu du fichier.sql qui se trouve dans le répertoire courant.

- **START fichier** Ouvre et lance le fichier.sql

- **QUIT ou EXIT** quitter sqlplus

Gestion du Buffer

R exécute une requête
L Liste le contenue du Buffer
L* Liste la ligne Courante
Ln liste la ligne numéro n
I insère une ligne après la ligne courante
A « texte » Ajoute texte après la ligne courante
DEL Supprime la ligne courante
CLEAR Efface le Buffer

Générer un fichier résultat appelé « spool »

- SPOOL OUT
- SPOOL fichier enregistre le buffer au fils de l'eau dans un fichier.
- SPOOL OFF met fin à cet enregistrement

Qu'est ce qu'un Script SQL ?

Un script sql, se présente comme un fichier texte, et dans ce cas respecte une syntaxe sql spécifique à oracle.

Poser un commentaire dans un fichier où requête SQL

* --commentaire sur une seule ligne
* /* commentaires sur plusieurs lignes*/

LDD Langage de Description des Données

Création de tables

CREATE TABLE [schéma.]nomtable
(
colonne1 type1 [DEFAULT valeur1] [NOT NULL]
[,colonne2 type2 [DEFAULT valeur2] [NOT NULL]]
[CONSTRAINT nomContrainte1 TypeContrainte1]…
) ;

- Schéma peut être omis, il sera donc assimilé au nom de l'utilisateur.

- Type des colonnes

Caractères (CHAR,NCHAR,VARCHAR2, NVARCHAR2,CLOB,NCLOB)
CHAR(n) & NCHAR(n) sont des chaînes de caractères de longueur fixe.
VARCHAR2 & NVARCHAR2 sont des chaînes de longueur variable.
CLOB & NCLOB sont des types acceptants des grands flux de caractères.

Valeur Numérique (NUMBER)
NUMBER permet de stocker des valeurs entières et flottante, on peut préciser le nb de chiffres après la virgule comme ceci NUMBER(n,d) ou N+D<=28

Date/heure (DATE,TIMESTAMP)

Le type DATE, permet d'exprimer des dates et heures JJMMYYYY HH:MM:SS
Le type TIMESTAMP permet d'exprimer des dates et heures avec une précision sur les millièmes de secondes.

Données binaires(BLOB,BFILE)
Le type BLOB permet de stocker des flux d'octets comme les images, la vidéo, le son.
Le type BFILE permet de stocker un pointeur sur un fichiers.

Utilisation :

CHAR(n) chaîne de caractères fixes.
VARCHAR2(n) chaîne de caractère variable.

NCHAR(n)
NVARCHAR2(n)

NUMBER((t,d)) valeur numérique de t chiffres dont d décimales.
INTEGER(t) <=> NUMBER(t,0)
DATE Permet de stocker des moments ponctuels, la précision est composée du siècle, de l'année, du mois, du jour, de l'heure, des minutes et des secondes.

- Les Contraintes

Les contraintes peuvent s'appliquer de 2 manières
Des contraintes sur les colonnes.
Des contraintes sur la table.

Les contraintes sur les colonnes sont

DEFAULT *valeur*
NOT NULL
UNIQUE *colonne*
CHECK (*condition*)

4 types de contraintes :

CONSTRAINT nomContrainte
UNIQUE (colonne1 [,colonne2]…)
PRIMARY KEY (colonne1 [,colonne2]…)
FOREIGN KEY (colonne1 [,colonne2]…)
REFERENCES [schéma.]nomtablepere (Colonne1 [,Colonne2]…)
[ON DELETE {CASCADE|SET NULL}]
CHECK (condition)

Exemple :

```
CREATE TABLE clients
(Idcli NUMBER NOT NULL,
 nom VARCHAR2(50),
prenom VARCHAR2(50),
age NUMBER NOT NULL,
idadr as NUMBER,
CONSTRAINTS pk_client PRIMARY KEY (IdCli),
CONSTRAINTS fk_client_adresse FOREIGN KEY (IdAdr) REFERENCES Adresse (IdAdr),
CONSTRAINTS ck_client_age CHECK (age BETWEEN 18 AND 100)
) ;
```

Attention : Il est fortement recommandé de nommer les contraintes.
pk_ fk_ ck_ uk_

Ajouter une contrainte :

ALTER TABLE NomTable
ADD (
CONSTRAINT nomContrainte
UNIQUE (colonne1 [,colonne2]…)
PRIMARY KEY (colonne1 [,colonne2]…)
FOREIGN KEY (colonne1 [,colonne2]…)
REFERENCES [schéma.]nomtablepere (Colonne1 [,Colonne2]…)
[ON DELETE {CASCADE|SET NULL}]
CHECK (condition)
) ;

Supprimer une contrainte :

ALTER TABLE NomTable
DROP CONSTRAINT **nomContrainte** [CASCADE] ;

Désactiver une contrainte :

ALTER TABLE DISABLE nomcontrainte ;

Activer une contrainte :

ALTER TABLE ENABLE nomcontrainte ;

Lister les contraintes posées :

SELECT * FROM USER_CONSTRAINTS ;

Obtenir les noms et descriptions des tables
Obtenir les noms des tables

SELECT TABLE_NAME **FROM** USER_TABLES ;

Obtenir la description de la table:

DESC nomtable ;

Permet d'obtenir la structure de la table dont le nom est indiqué.

Poser un commentaire sur une table ou une colonne d'une table en SQL
COMMENT ON TABLE [schéma.]NomTable IS 'MON COMMENTAIRE' ;

COMMENT ON COLUMN [schéma.]NomTable.nomColonne IS 'MON COMMENTAIRE' ;

Voir les commentaires posés :

SELECT * FROM USER_TAB_COMMENTS ;

Supprimer une table
DROP TABLE [schéma.]NomTable [CASCADE CONSTRAINTS] ;

CASCADE CONSTRAINTS permet d'indiquer la suppression des contraintes associées à cette table.

Renommer une table
RENAME ancienNom TO nouveauNom ;

Les contraintes d'intégrité, index et prérogatives associées à l'ancienne table sont automatiquement transférées sur la nouvelle.

Attention : Les vues, synonymes, procédures catalogués sont inchangés, il faut donc les recréer.

Autre manière de renommer une table
ALTER TABLE noTable RENAME TO NouveauNom ;

Ajouter une colonne
ALTER TABLE NomTable ADD NomColonne TypeCol CONSTRAINTS nomContraintes typeContraintes ;

Renommer colonne
ALTER TABLE NomTable RENAME COLUMN Nomcolonne TO Nouveau NomColonne ;

Modifier type des colonnes
ALTER TABLE Nomtable MODIFY Nomcolonne TypeColonne Contraintes ;

Exemple

ALTER TABLE piste **MODIFY** nom CHAR(10) **NOT NULL** ;

Supprimer des colonnes
ALTER TABLE nomTable DROP COLUMN nomColonne ;

Attention : Pas toutes les versions, ne supprime pas les colonnes comportant une clé primaire, ne supprime pas les colonnes utilisées dans les index, et ne peut pas supprimer toutes les colonnes.

LMD Langage de Manipulation des Données

Création des séquences

Les séquences sont des objets qui permettent de créer des séquences de valeur avec incrément ou décrément automatique.
Créer une séquence :

CREATE SEQUENCE nomSequence
MINVALUE valeurMin | **NOMINVALUE**
MAXVALUE valeurMax | **NOMAXVALUE**
START WITH valeurDeDebut
INCREMENT BY valeurIncrément
CYCLE|NOCYCLE
CACHE n | NOCACHE

ORDER | NOORDER;

Les valeurs par défauts sont :
NOMINVALUE => 1
NOMAXVALUE => $2^{32} - 1$
START WITH =1
INCREMENT BY = 1
CYCLE
CACHE= 20
ORDER

Modifier une séquence :

ALTER SEQUENCE nomSequence
MINVALUE valeurMin | **NOMINVALUE**
MAXVALUE valeurMax | **NOMAXVALUE**
START WITH valeurDeDebut
INCREMENT BY valeurIncrément
CYCLE|NOCYCLE
CACHE n | NOCACHE
ORDER | NOORDER;

Supprimer une séquence :

DROP SEQUENCE NomSequence ;

Pour faire appel a une séquence ou donne son nom suivie de la commande NEXTVAL.

Il est également possible d'utiliser la commande CURRVAL qui permet d'obtenir la valeur courante du compteur sans incrémenter celui ci (lecteur seule).

L'utilisation pour la première fois de NEXTVAL pour un compteur donne la valeur courante de celui ci, les utilisation suivantes donnent les valeurs suivantes (fonction incrémentation activé).

TestSequence.NEXTVAL

Info :

La table **DUAL**, est une table particulière, elle ne possède qu'un champs et une valeur(X)

Cette table DUAL peut être utilisé pour tester des formules, des séquences ou envoyer vers la sortie résultats du texte.

Exemple :

SELECT "bonjour" FROM DUAL ; => bonjour

SELECT 3+2 FROM DUAL ; => 5

On peut aussi tester les séquences avec la table DUAL

SELECT seq.CURRVAL FROM DUAL ; => 1

SELECT seq.NEXTVAL FROM DUAL ; => 2

SELECT seq.NEXTVAL FROM DUAL ; => 3

Insérer des données dans vos tables

Pour créer une ou plusieurs lignes dans une table, SQL offre l'instruction INSERT INTO.

INSERT INTO nomtable(colonne1,colonne2,….) **VALUES**(valeur1,valeur2,….) ;

Plusieurs précautions à prendre :

- Les valeurs données dans le VALUES doivent être dans le même ordre et le même type que ceux de la colonne.

- Toutes les colonnes qui ne sont pas précisées, reçoivent alors la valeur par défaut. (Souvent NULL ou celle défini par la contrainte DEFAULT valeur1)

- Si la moindre valeur ne vérifie pas les contraintes de la table, alors l'instruction INSERT INTO échoue.

Exemple : INSERT INTO clients(numero,nom,prenom,code_postal)
VALUES ('120','Tondeur','Hervé','59300') ;

Exemple d'utilisation des séquence :

***CREATE SEQUENCE** AutoInc*
***MINVALUE** 1*
***MAXVALUE** 999999*
***START WITH** 1*
***INCREMENT BY** 1 ;*

***INSERT INTO** clients(numero,nom,prenom,code_postal)*
***VALUES** (AutoInc.NEXTVAL,'Tondeur','Hervé','59300') ;*

***INSERT INTO** MaTable(Id,Nom,Prenom,DDN) **VALUES**(MaSeq.NEXTVAL,'Tondeur','Hervé','12/01/1980') ;*

Utilisation de sous requête pour l'insertion de données dans un table

Il est possible d'utiliser une sous requête qui renvoie un ensemble de tuples, qui pourront être réintégré dans votre table (souvent utilisé pour ma migration de données).

Dans ce cas particulier on n'utilise pas de clause VALUE, elle est implicite.

*Exemple : **INSERT INTO** commandes(numero,jour,client)*
***SELECT** numero,jour,client*
***FROM** commandes_anciennes*
***WHERE** (SYSDATE-jour)>=180 ;*

Modifier des données dans vos tables

SQL vous permet de modifier une ou plusieurs données de vos tables grâce à la commande :

UPDATE nomTable **SET** colonne='valeur' [**WHERE** condition] ;

*Exemple : **UPDATE** articles **SET** prix=prix/6.55957 ;*

*Exemple : **UPDATE** articles **SET** prix=prix/6.55957 **WHERE** devise='EURO' ;*

Lorsque la clause conditionnelle est omise, toutes les lignes de la colonne sont mise à jour.
Il est possible de mettre plusieurs colonne à jour en même temps, il faut se méfier des corrélations entre colonnes.

Quelques fonctions de conversions utiles sur les dates/heures

TO_CHAR(colonneDate, format) convertir une date en chaîne de caractères.

*SELECT **TO_CHAR**(begin_time,'DD/MM HH24:MI') begin_time,**TO_CHAR**(end_time,'DD/MM HH24:MI') end_time, undoblks, maxquerylen FROM v$undostat ;*

TO_DATE(ChaîneCaractéres, format) convertir une chaîne en type Date.
*UPDATE commande SET dateCmd=**TO_DATE**('24/04/2016 13:13:00','DD/MM/YYYY HH24:MI:SS') WHERE IdCmd='3' ;*

Tableau des codes de format utilisables avec TO_DATE et TO_CHAR

SCC	Siècle (avec signe)
CC	Siècle
SY, YYY	Année (avec signe et virgule)
Y,YYY	Année (avec virgule)
YYYY	Année sur 4 chiffres
YYY	Les 3 derniers chiffres de l'année
YY	Les 2 derniers chiffres de l'année
Y	Le dernier chiffre de l'année
Q	Numéro du trimestre dans l'année
WW	Numéro de semaine dans l'année
W	Numéro de semaine dans le mois
MM	Numéro du mois
DDD	Numéro du jour dans l'année
DD	Numéro du jour dans le mois
D	Numéro du jour dans la semaine

HH	Heure sur 12 heures
HH24	Heure sur 24 heures
MI	Minutes
SS	Secondes
J	Jour du calendrier julien
YEAR	Année en toute lettres
MONTH	Nom du mois
MON	Nom du mois abrégé sur les 3 premières lettres
DAY	Nom du jour
DY	Nom du jour abrégé sur les 3 premières lettres
AM	Indication AM
PM	Indication PM
BC	Indication BC
AD	Indication AD
TH	Ajout du suffixe ordinal ST, ND, RD, TH au nombre considéré
SP	Ecriture en toutes lettres du nombre considéré
RR	Deux derniers chiffres de l'année en cours

EXTRACT({YEAR|MONTH|DAY|HOUR|MINUTE|SECOND} FROM {Expression-Date| Expression-Interval}) extrait une parte données d'une date ou d'un intervalle.

NUMTOYMINTERVAL(expressionnumerique,{'YEAR'|'MONTH'}) convertir un nombre dans un type INTERVAL YEAR TO MONTH

NUMTODSINTERVAL(expressionnumerique,{'DAY'|'HOUR'|'MINUTE'|'SECOND'}) convertir un nombre dans un type INTERVAL DAY TO SECOND.

Supprimer des données dans vos tables

Pour supprimer une ou plusieurs ligne dans un seule table, SQL prévoit l'instruction :

DELETE FROM nomTable [**WHERE** condition] ;

Exemple : DELETE FROM clients ;

Supprimer tous les clients

***DELETE FROM** commandes **WHERE** (SYSDATE – jour)>=180 ;*

NB : Si une des lignes est référencé par une clé étrangère alors la requête est refusée.

Attention l'utilisation de la clause DELETE sans clause WHERE est dangereuse, il est recommandé d'utiliser la clause TRUNCATE dans le cas où on désire supprimer l'ensemble des lignes d'une table.

TRUNCATE TABLE NomTable **[{DROP|REUSE} STORAGE] ;**

L'option **DROP STORAGE**, permet de supprimer l'espace disque occupé par ces données qui vont être supprimées.
L'option **REUSE STORAGE**, permet de conserver l'espace disque occupé par ces données.

Sélection de données

Pour afficher les données sous forme d'un tableau dont les colonnes porte un intitulé, SQL propose l'instruction :

SELECT colonnes **FROM** NomTable **WHERE** Conditions ;
(1) (2)

1) **les colonnes**

- **Les Alias de tables et de colonnes**

Les alias de tables, permettent de préfixer les tables, il est fortement recommandé par Oracle d'utiliser les alias de tables.
Les alias de tables sont obligatoires quand deux tables portent le même nom, comme par exemple dans l'auto-jointure, ou dans l'appel de base de données externe avec un database Link.

Exemple :
SELECT c.Nom, c.Prenom FROM CLIENTS c ;

Les alias de colonnes permettent de renommer des colonnes à l'affichage.
Les alias de colonnes sont utiles pour les calculs.

On utilise le mot clef AS

Exemple :

***SELECT** compa AS C1, nom **AS** NomEtPrenom, brevet C3 **FROM** pilote ;*
***SELECT** pl.compa **AS** C1, pl.nom **AS** NomEtPrenom, pl.brevet **AS** C3 FROM pilote **AS** pl ;*

NB : Oracle traduit les noms des alias en majuscules.
L'utilisation du mot clef AS est facultative
Il faut préfixer les colonnes par l'alias de la table lorsqu'il existe.

- **les colonnes et le caractères joker ***

Il est possible dans le nom des colonnes d'utiliser le caractère spécial * qui permet de nommer toutes les colonnes d'une ou plusieurs tables.

- **Les colonnes calculées**

Dans la clause SELECT, on peut utiliser les opérateurs Arithmétiques(+ - * /), l'opérateur de concaténation des chaînes de caractères (||) ou des fonctions SQL (comme MOD pour le modulo, FLOOR pour la troncature entière ou SUBSTR pour l'extraction de sous chaînes de caractères)

Dés que l'on à un doute sur les priorités des opérateurs, il faut utiliser les parenthèses afin de lever toute ambiguïté.

A partir du moment ou l'on utilise des colonnes calculées, il est conseillé de lui donner un nom (un alias) avec le mot clef AS pour en expliquer le contenue.
Un alias est donné entre double quotes « prixTTC »

Exemple : ***SELECT*** *SUBSTR(prenom,1,1) || '.' Nom* ***AS*** *« identité »,*
FLOOR ((SYSDATE – naissance) / 365.25) ***AS*** *« Age »,*
MOD((SYSDATE – naissance), 365) ***AS*** *« Nb de jour depuis l'anniversaire »*
FROM *clients ;*

- **Les fonctions d'agrégation SQL**

Syntaxe	Description
COUNT	Dénombrement des lignes
SUM	Somme des valeurs numériques
MIN	Valeur Numérique Minimale
MAX	Valeur Numérique Maximale
AVG	Moyenne des valeurs numériques

Ces fonctions ignorent les valeurs NULL dans leur calcul et n'éliminent pas les doublons.

Exemple :

SELECT *MAX(prix)* ***AS*** *« prix le plus cher »,*
MIN(prix) ***AS*** *« Prix le moins cher »,*
AVG(prix) ***AS*** *« prix moyen »*
FROM *articles ;*

SELECT COUNT*(numero)* ***AS*** *« Nombre de clients »*
FROM *clients ;*

- **quelques fonctions Oracles**

ASCII(c) Retourne la valeur du caractère ASCII équivalent.
CHR(n) Retourne le caractère ASCII équivalent
CONCAT (c1,c2) Équivalent à ||
INITCAP (c) Première lettre de chaque mot mis en majuscule
UPPER (c) Met en majuscule la chaîne
LOWER(c) Met en minuscule la chaîne
LENGTH(c) Longueur de la chaîne
REPLACE(c1,c2,c3) recherche c2 dans c1 et le remplace par c3

TRIM(c) supprime les espaces devant et derrière la chaîne.

ABS(n) valeur absolue
CEIL(n) plus petit entier
FLOOR(n) plus grand entier
ROUND(n,d) arrondi a d décimale la valeur n
SIGN(n) signe de n
TRUNC(n,d) coupure de n à d décimales

SYSDATE Date courante
LAST_DAY(d) retourne le dernier jour du mois

GREATEST(exp1,exp2,exp3) retourne la plus grande des expressions
LEAST(exp1,exp2,exp3) retourne la plus petite des expression
NULLIF(exp1,exp2) retour NULL si exp1=exp2 sinon Exp1
NVL(exp1,exp2) converti exp1 en exp2 si exp1=NULL

2) La clause WHERE

La clause WHERE peut être très riche en conditions de sélection

- **Les opérateurs de comparaisons**

  < = > <= >= <>
  SELECT * FROM articles WHERE prix<100 ;

- **Comparateur de vacuité**

  La Syntaxe =NULL ou <>NULL renvoie toujours « faux », on ne peut tester la vacuité d'une colonne qu'avec la syntaxe IS [NOT] NULL

  *Exemple SELECT * FROM clients WHERE cp IS NOT NULL ;*

- **Les opérateurs Logiques**

  Les mots clés **AND** et **OR** sont mis à disposition

  *Exemple **SELECT * FROM** articles **WHERE** couleur='bleu' **AND** (prix>100 **OR** prix<10) ;*

- **Les bornes d'inclusions**

  L'opérateur BETWEEN permet de remplacer avantageusement la séquence
  >= …. AND …. <=

  *Exemple : **SELECT * FROM** articles **WHERE** prix **BETWEEN** 10 **AND** 100 ;*

- **Sélection dans une liste**

  L 'opérateur IN permet vérifier qu'une valeur est contenue dans une liste de valeurs.
  Nb : il peut être associé à l'opérateur NOT pour signifier n'est pas dans.

  *Exemple : **SELECT * FROM** clients **WHERE** nom **NOT IN** ('Dupont','Durand') ;*

- **Comparateur de Motifs**

  Il est possible de balayer une colonne de type chaîne de caractères à la recherche d'un motif, grâce à l'opérateur de filtre LIKE et du caractère % qui remplace toute série de caractère(s) (y compris vide).

  Exemple : ***SELECT * FROM*** *clients* ***WHERE*** *nom* ***LIKE*** *'D%' ;*
  SELECT * FROM *clients* ***WHERE*** *nom* ***LIKE*** *'%e' ;*

  Il existe également le caractère _ qui remplace 1 caractère, on peut l'utiliser plusieurs fois de suite.

- **Opérations sur le résultat**

  Le résultat d'une requête SELECT peut comporter plusieurs fois la même ligne. Pour éliminer les lignes doublons, il existe les mot-clés DISTINCT, UNIQUE, ALL

  DISTINCT et UNIQUE permettent d'éliminer les éventuels duplicatas.

  ALL affiche l'ensemble des données (c'est l valeur par défaut).

  Exemple : ***SELECT DISTINCT*** *nom* ***FROM*** *clients ;*

  - **Opérateur de trie**

  Il est également possible de trier les lignes du résultat, pour cela nous disposons de la clause ORDER BY.

  Exemple : ***SELECT * FROM*** *clients* ***ORDER BY*** *nom* ***ASC NULLS FIRST*** *;*

  On utilise les mots clés :

  ASC pour classer dans l'ordre croissant
  DESC pour classe dans l'ordre décroissant
  NULLS FIRST les valeurs nulles en haut
  NULLS LAST les valeurs nulles en bas

  On peut utiliser dans la clause ORDER BY les alias de la clause SELECT

  La zone de trie, utilise la SORT AREA de la PGA, cette SORT AREA est parametrable grâce à l'option SORT_AREA_SIZE du fichier init.ora.

  - **Le champ ROWNUM**

  Enfin pour garder que les x premières lignes du résultat, Oracle à prévu de numéroter les lignes du résultat dans un colonne cachée nommée ROWNUM, qui se trouve dans vos tables propriétaire.

  Exemple : ***SELECT DISTINCT*** *nom* ***FROM*** *clients* ***WHERE*** *ROWNUM<=100*
  ORDER BY *nom* ***ASC NULLS LAST*** *;*

Les opérations ensemblistes

On peut articuler les résultats de plusieurs requêtes homogènes à l'aide des opérations ensemblistes UNION, INTERSECT et MINUS.

Les règles à respecter avec ces opérations ensemblistes sont les suivantes :

- Les colonnes affichées par les deux requêtes doivent être compatibles, en nombre , en ordre et en type de données.

- Les éventuels alias ne sont définis que dans la première clause SELECT.

- Une éventuelle clause ORDER BY n'es possible qu 'à la fin de la dernière requête, car elle agit sur le résultat final.

Dernières remarques :

- Par défaut, l'opération UNION élimine les doublons ; pour les conserver, il faut utiliser l'opérateur UNION ALL.

- Contrairement à UNION et INTERSECT, l'opérateur MINUS n'est pas commutatif, donc l'ordre dans lequel les requêtes sont écrites, de part et d'autre, a de l'importance.

Exemple :

```
-- affiche les identité de tous les clients et de tous les employés
SELECT prenom || ' ' || nom AS « identité » FROM clients
UNION
SELECT prenom || ' ' || nom AS « identité » FROM employes ORDER BY « identité » ASC ;

--Idem--
SELECT prenom || ' ' || nom AS « identité » FROM employes
UNION
SELECT prenom || ' ' || nom AS « identité » FROM clients ORDER BY « identité » ASC ;

-- affiche les identités de tous les clients qui ont un homonyme parmi les employés
SELECT prenom || ' ' || nom AS « identité » FROM clients
INTERSECT
SELECT prenom || ' ' || nom AS « identité » FROM employes ORDER BY « identité » ASC ;

--Idem--
SELECT prenom || ' ' || nom AS « identité » FROM employes
INTERSECT
SELECT prenom || ' ' || nom AS « identité » FROM clients ORDER BY « identité » ASC ;

-- affiche les identités de tous les clients qui n'ont pas d'homonyme parmi les employés
SELECT prenom || ' ' || nom AS « identité » FROM clients
MINUS
SELECT prenom || ' ' || nom AS « identité » FROM employes ORDER BY « identité » ASC ;

-- affiche les identités de tous les employés qui n'ont pas d'homonyme parmi les clients
SELECT prenom || ' ' || nom AS « identité » FROM employes
MINUS
SELECT prenom || ' ' || nom AS « identité » FROM clients ORDER BY « identité » ASC ;
```

Les Jointures

Equi-Jointure

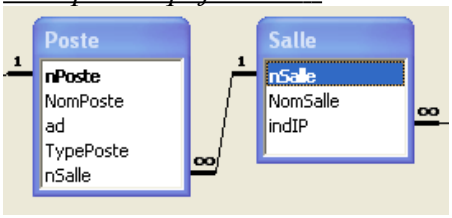
Une equi-jointure utilise l'opérateur = et compare en généralement des clés primaires avec des clés secondaires.

Oracle recommande d'utiliser des alias de tables pour améliorer les performances.

Les alias sont obligatoires pour des colonnes qui portent le même nom ou pour les auto jointures.

On va trouver deux formes d'écritures, relationnelle (=) et SQL2 (JOIN…ON condition), l'utilisation de INNER devant JOIN est optionnelle et est appliqué par défaut.

Exemple d'équijointure :



SELECT *NomSalle, NomPoste, TypePoste*
FROM *Poste,Salle*
WHERE *Poste.nSalle=Salle.nSalle* ***AND*** *Salle.NomSalle='07T' ;*

SELECT *NomSalle, NomPoste, TypePoste*
FROM *Salle* ***JOIN*** *Poste* ***ON*** *Poste.nSalle=Salle.nSalle*
WHERE *Salle.NomSalle='07T' ;*

Auto Jointure

Cas particulier de l'équijointure, l'auto jointure, relie une table à elle même.

Soit la table suivante :

Commandes(**numero**,client, jour)

Avec une autojointure il est possible de répondre à la question suivante : Afficher les numéros des clients qui ont commandé en même temps.

SELECT DISTINCT *a.client, b.client*
FROM *commandes a, commandes b*
WHERE *a.client < b.client*
AND *a.jour=b.jour ;*

SELECT DISTINCT *a.client, b.client*

***FROM** commandes a **JOIN** commandes b **ON** a.client<b.client*
***WHERE** a.jour=b.jour ;*

DISTINCT est obligatoire car sinon, deux clients qui ont commandé en même temps deux fois, apparaîtraient deux fois.

Jointure externe

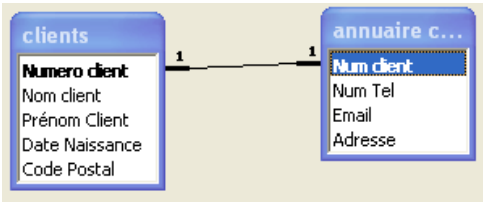
Les jointures externes permettent d'extraire des enregistrements qui ne répondent pas aux critères de jointure. Lorsque deux tables sont en jointure externe, une tables est « dominante » par rapport à l'autre qui est dite « subordonnée ». Ce sont les enregistrements de la table dominante qui sont retournés (même s'il ne satisfont pas aux conditions de jointure).

Ecriture relationnelle

- La directive de jointure externe (+) se place du côté de la table subordonnée.
- Cette directive peut se placer à gauche ou à droite d'une clause de jointure, pas des deux côtés.
- Une clause de jointure externe ne peut ni utiliser l'opérateur IN ni être associé à une autre condition par l'opérateur OR.

Ecriture SQL2

- Le sens de la directive de jointure externe LEFT ou RIGHT de la clause OUTER JOIN désigne la table dominante.



--Affichage de toutes les informations relatives aux clients dont on connaît les informations complémentaires.

***SELECT** **
***FROM** clients a, annuaire_client b*
***WHERE** a.numero_client=b.num_client ;*

Cette requête n'affiche pas tous les clients car il existe des clients pour lesquels on ne connaît aucune information complémentaire.

Il faut donc que la condition de jointure ne soit pas obligatoire, afin de ne pas supprimer à l'affichage les clients dont on n'a pas d'information complémentaire.

On parle alors de jointure externe. Il suffit pour cela d'ajouter le signe (+) dans la condition de jointure du coté de la table subordonnée :

Ecriture relationnelle

```
SELECT *
FROM clients a, annuaire_client b
WHERE a.numero_client=b.num_client (+) ;
```

Une requête équivalente

```
SELECT *
FROM clients a, annuaire_client b
WHERE b.num_client (+) = a.numero_client ;
```

Ecriture SQL2

```
SELECT *
FROM clients a LEFT OUTER JOIN annuaire_client b ON
(a.numero_client=b.num_client);
```

Une requête équivalente

```
SELECT *
FROM annuaire_client b RIGHT OUTER JOIN clients a ON
(a.numero_client=b.num_client);
```

Les Sous Requêtes ou jointures procédurales

Il est possible d'utiliser dans une requête principale, une sous requête SELECT à condition que cette dernière soit délimitée par des parenthèses.

Sous requête qui renvoient une colonne et une seule valeur

Lorsque la ss-requête renvoie de manière sure une seule valeur, il est possible d'utiliser les opérateurs de comparaisons classique.

Exemple :

```
SELECT * FROM articles
WHERE prix >= (SELECT MAX(prix) FROM articles WHERE couleur='bleu') ;
```

Ces ss-requête peuvent également s'employer avec des opérateurs arithmétiques.

Exemple :

```
SELECT
(
SELECT COUNT(*)
FROM clients
WHERE FLOOR((SYSDATE – b.naissance) /365.25) <18
)
/
(
```

```
SELECT COUNT(*)
FROM clients
) * 100 AS « proportion de clients mineurs »
FROM DUAL
```

Sous requête qui renvoient une colonne et plusieurs valeurs

Lorsque la sous requête renvoie une colonne de valeurs, la requête principale peut utiliser un opérateur de comparaison classique, mais accompagné soit de ALL soit de ANY et IN :

Exemple :

```
SELECT * FROM articles
WHERE prix >= ANY (SELECT prix
                   FROM articles
                   WHERE couleur='bleu') ;
```

ALL ⇔ a tous
ANY ⇔ au moins un

IN ⇔ = ANY

NOT IN ⇔ <>ALL

Sous requête qui renvoie plusieurs colonnes et plusieurs lignes

Une sous requête peut renvoyer plusieurs colonnes de valeurs, auquel cas elle peut être utilisée dans la clause FROM de la requête principale.

Exemple :

```
SELECT *, FLOOR(SYSDATE – b.naissance) / 365.25) AS « Age »
FROM (SELECT *
      FROM clients
      WHERE ROWNUM <=10
      ORDER BY naissance DESC) a , annuaire_client b
WHERE a.client = b.numero (+) ;
```

Une sous requête multi colonnes est également utilisable avec un opérateur de comparaison entre couples et tous les autres tuples.

Corrélation

Une sous requête peut avoir besoin d'information provenant de la requête principale. On dit alors que la requête est corrélée et le passage d'information se fais par l'utilisation d'alias.

Attention avec la corrélation il risque d'y avoir beaucoup de calcul et de pertes de temps, préférer utiliser une technique différente si possible.

Autres Utilisations

L'emploie d'une sous requête peut être appelé également par une requête INSERT et également dans une requête DELETE

Les clauses de Groupements

Il est parfois nécessaire de calculer la même fonction d'agrégation sur plusieurs de lignes.

La clause GROUP BY

Exemple pour calculer le montant total de chaque commande, il est nécessaire de grouper les lignes de commandes.

Exemple :

```
SELECT a.commande, SUM(a.quantité * b.prix) AS « Montant total »
FROM ligne_commande a , articles b
WHERE a.article = b.numéro
GROUP BY a.commande ;
```

Pour reconnaître une requête SELECT qui nécessite une clause GROUP BY, il suffit de se demander si le résultat doit afficher plusieurs valeurs d'une même fonction d'agrégat.

Attention : il existe deux contraintes impératives sur les colonnes de la clause SELECT ;
Toutes les colonnes de la clause GROUP BY doivent figurer sur la clause GROUP BY
Toutes les colonnes de la clause SELECT sans fonction d'agrégat, doivent figurer dans la clause GROUP BY

Il est parfois indispensable de procéder par plusieurs vue préliminaires avant de pouvoir calculer.

Exemple :

```
--calcul des revenus
CREATE OR REPLACE VIEW revenus (formation, revenu)
AS
SELECT a.formation, SUM(a.présence * b.frais)
FROM participations a, formations b
WHERE a.formation=b.numero
GROUP BY a.formation ;

--calcul des marges
SELECT a.formation, a.revenu – b.coût AS « profit »
FROM revenus a, couts b
WHERE a.formation=b.formation ;

--calcul des coûts
CREATE OR REPLACE VIEW coûts (formation, coût)
AS
SELECT a.formation, SUM(a.heures * b.bonus * c.tarifs + b.prime + b.fixe)
FROM animation a, formations b, animateurs c
WHERE a.formation=b.numero AND a.animateur=c.numero
GROUP BY a.formation ;
```

Regroupement sur plusieurs colonnes

La clause GROUP BY permet d'effectuer des groupements sur plusieurs colonnes (puis des calculs d'agrégation sur ces sous-groupes).

Exemple :

```
SELECT a.client, b.article, MAX(a.quantité) AS « Quantité MAX »
FROM commandes a , ligne_commandes b
WHERE a.numero=b.commande
GROUP BY a.client, b.article ;
```

L'ordre des colonnes dans la clause GROUP BY n'a pas d'importance et l'ordre des mêmes colonnes dans la clause SELECT n'a pas besoin d'être le même.

Par contre il est conseillé d'utiliser la clause ORDER BY pour laquelle l'ordre à de l'importance.

Exemple :

```
--Pour afficher le résultat d'abord par N° client croissant puis
-- (à l'intérieur d'un même N° client) par N° article décroissant
SELECT a.client,b.article, MAX(a.quantite) AS « Quantite MAX »
FROM commandes a, lignes_commandes b
WHERE a.numero=b.commande
GROUP BY a.client, b.article
ORDRE BY a.client ASC, b.article DESC ;
```

La clause HAVING

Il est parfois utilise d'exclure certains groupes issus d'une requête GROUP BY à l'aide d'une ou plusieurs conditions de sélection.
Ces conditions ne peuvent être écrites dans la clause WHERE qui est réservée à l'exclusion des lignes avant groupement, mais dans une dernière clause, la clause HAVING.

Exemple :

```
SELECT client, COUNT(numero) AS « Nombre de commandes »
FROM commandes
GROUP BY client
HAVING COUNT(numero)>=4 ;
```

Les alias ne peuvent être utilisé dans une clause HAVING, il faut en conséquence de quoi répéter la formule.

Tous les opérateurs qui sont autorisé dans une clause WHERE peuvent être utilisés dans la clause HAVING, y compris une sous requête.

Par contre les conditions de sélection avant groupement (et les conditions de jointure) doivent rester dans la clause WHERE.

Exemple :

```
SELECT a.article, b.client, SUM(a.quantite) AS « quantite totale »
FROM ligne_commandes a, commandes b
WHERE a.commande=b.numero
GROUP BY a.article, b.client
HAVING (a.article, SUM(a.quantite)) IN (SELECT article, MAX(quantite_totale)
                                        FROM QuantitesTotales
                                        GROUP BY client) ;
```

La clause WHERE peut porter sur les colonnes de la clause SELECT sans fonction d'agrégation ainsi que sur les colonnes non affichées.

La clause HAVING ne peut porter que sur les colonnes de la clause SELECT (agrégées ou non).

Créer des Vues :

- On peut donner un nom à une requête, puis l'utiliser comme une table dans une autre requête.

- Une vue est une table virtuelle qui permet de regrouper des champs de plusieurs table en une seule.

- De simplifier les requêtes SQL.

- D'apporter une sécurité dans la manipulation des données qui se font via les vue et non plus directement à partir des tables.

- De permettre de montrer aux utilisateurs que les données qui les concernent et de cachés les schémas relationnels.

- De donner des intitulés aux colonnes plus parlant que ceux des tables, et parfois de traduire dans différentes langue ou langages métiers.

On utilise pour cela la clause SQL suivante :

CREATE OR REPLACE VIEW nomVue
(nomcolvue1,nomcolvue2,….)
AS
SELECT col1,col2, ….
FROM Nomtables
WHERE conditions
[WITH READ ONLY] ;

L'option **WITH READ ONLY**, rend la vue non modifiable.

Les nouveaux intitulés de colonnes donnés lors de la définition de la vue pourront être réutilisés comme n'importe quel nom de colonne.

Les Vues permettent d'effectuer des requêtes INSERT, DELETE et UPDATE sur plusieurs tables à la fois, à conditions d'utiliser des déclencheurs ou d'avoir une vue modifiable.

Une vue est modifiable si et seulement si, on n'utilise pas les mots clefs suivants dans la conception de la vue :
DISTINCT
AVG, COUNT, MAX, MIN, VARIANCE, STDDEV
ROWNUM, ROWID, LEVEL
GROUP BY
HAVING
ORDER BY
CONNECT BY

Supprimer une vue :

DROP VIEW NomVue **[CASCADE CONSTRAINTS] ;**
L'option **CASCADE CONSTRAINTS**, permet de supprimer les clés primaire et étrangéres.

Créer des Synonymes :

CREATE OR REPLACE [PUBLIC] SYNONYME NomSynonyme
FOR NomObjetOracle[@Database Link] ;

L'option **PUBLIC**, permet de créer un synonyme public accessible par tous.
NomObjetOracle, représente un objet d'un schéma Oracle, comme une table par exemple.

DROP [PUBLIC] SYNONYME NomSynonyme **[FORCE]** ;

L'option **FORCE**, force la suppression si le synonyme est dépendant d'un autre objet, comme une vue par exemple.

Créer des Databases Link :

CREATE [PUBLIC] DATABASE LINK NomDbLink
CONNECT TO <utilisateur>
IDENTIFIED BY <Password>
USING <schéma> ;

Permet de créer une connexion avec une base de données distante, et de l'interroger comme si cette base de données était locale.

Exemple :
CREATE DATABASE LINK ComptaMaubeuge
CONNECT TO System
IDENTIFIED BY manager
USING CPTMAUBEUGE ;

SELECT * FROM Ecritures@ComptaMaubeuge ;

DROP DATABASE LINK NomDbLink **[FORCE]** ;

L'option FORCE, permet de forcer la suppression du database link, même si celui ci est verrouiller par un autre objet Oracle.

Lister l'ensemble des DATABASE LINK du schéma Oracle :
SELECT * FROM DBA_DB_LINKS ;

Accélérer les recherche grâce aux Index :

Un Index Oracle permet d'accélérer l'accès aux données d'une table.
Le but principal d'un index est d'éviter de parcourir une table séquentiellement du premier enregistrement jusqu'à celui visé.

Plusieurs type d'index sont proposés par Oracle :
- L'arbre équilibré (B-tree)
- Inverse (Reverse Key) qui concerne les tables « clustérisées ».
- Chaîne de bits (Bitmap) qui regroupe chaque valeur de la ou des colonnes indexées sous la forme d'une chaîne de bits.
- Basés sur les calculs entre colonnes (function-based indexes)

CREATE INDEX {UNIQUE|BITMAP} [schéma.]nomIndex
ON [schéma.]nomTable ({colonne1 | expressionCol1} [ASC | DESC] ….) ;

UNIQUE permet de créer un index qui ne supporte pas les doublons.
BITMAP fabrique un index « chaînes de bits »
ASC et DESC précisent l'ordre (croissant ou décroissant).

Exemple :

```
CREATE UNIQUE INDEX
Idx_compte
ON CompteEpargne (titulaire DESC) ;
```

Index B-tree ordre inverse

```
CREATE INDEX idx_debitFF
ON CompteEpargne (debit*6.55957) ;
```

Index B-tree expression d'une colonne

```
CREATE BITMAP INDEX
Idx_Bitmap_txInt_CompteEpargne
ON CompteEpargne (txInt) ;
```

Index Bitmap

```
CREATE INDEX
Idx_fct_Solde_CompteEpargne
ON CompteEpargne
((credit-Debit)*(1+(txInt/100))-agios,
credti,debit,txInt,agios) ;
```

Index basé sur une fonction

LE LCD (Langage de contrôle des données)

1. Notion d'utilisateur de rôle et de privilège

Chaque utilisateur d'une base de données (comme Oracle ou PostgreSQL par exemple) dispose d'un nom et d'un mot de passe, et possède également des tables, des vues et d'autres ressources qu'il a créées. Dans Oracle, un rôle représente un ensemble de privilèges. Vous pouvez assigner des privilèges spécifiques à des rôles, puis assigner ces rôles aux utilisateurs appropriés. Un utilisateur peut aussi attribuer des privilèges à d'autres utilisateurs s'il a été accrédité pour.

Deux types de privilèges :

•Privilège au niveau système : qui donne le droit d'exécuter une action particulière sur n'importe quel objet. Le privilège CREATE TABLE, par exemple, permet de créer des tables. Le privilège GRANT ANY PRIVILEGE permet d'accorder des privilèges à d'autres utilisateurs.
•Privilège au niveau objet : qui donne le droit d'exécuter une action donnée sur un objet

spécifique. Le privilège SELECT, par exemple, permet d'exécuter une opération SELECT sur une table, une vue, une séquence (ou un snapshot sous Oracle).

La norme SQL2 propose trois fonctions pour connaître l'utilisateur connecté : SYSTEM_USER (nom de l'utilisateur connecté), SESSION_USER (nom d'utilisateur qui a ouvert la session), CURRENT_USER (nom de l'utilisateur courant). On utilise ces fonctions avec une commande SELECT.

Créer un utilisateur

CREATE USER <utilisateur> IDENTIFIED BY <mot_de_passe> | EXTERNALLY

Le paramètre EXTERNALLY permet de ne pas définir de mot de passe si le mot de passe du système d'exploitation de l'ordinateur vous suffit pour identifier l'utilisateur. Stratégie dangereuse tout de même, car il vaut mieux avoir un autre filtre sur la base de donnée qui offre un second niveau de sécurité.

On peut également fixer pour cet utilisateur un espace disque spécifique (TABLESPACE) et une limitation de son utilisation des ressources (QUOTAS).

Modifier le mot de passe des utilisateurs

ALTER USER <utilisateur> IDENTIFIED BY <nouveau_mot_de_passe>

On peut gérer la « vie » des mots de passe grâce à la création de profils (CREATE PROFILE) qui permettent au DBA de fixer des conditions d'accès :

- durée de vie d'un mot de passe ;
- période de grâce qui suit l'expiration d'un mot de passe et pendant laquelle il peut être changé ;
- nombre d'échecs de connexion répétés autorisés avant de verrouiller le compte ;
- durée (en jours) de verrouillage d'un compte ;

- nombre de jours qui doivent s'écouler avant de pouvoir changer un mot de passe ;
- nombre de changements de mot de passe qui doivent avoir lieu avant de réutiliser un mot de passe.

Supprimer un utilisateur

DROP USER <utilisateur> [CASCADE]

2. Les rôles standards

Les rôles ont été implémentés dans le langage SQL à partir de 1999. Seules les bases de données solides comme Oracle ont implémenté ces ordres. Les rôles peuvent être apparentés à des groupes, ou des profils qui permettent de définir un ensemble de privilèges à attribuer à des utilisateurs de la base de données.

Il existe trois rôle par défaut dans Oracle :

•CONNECT : Pour les utilisateurs occasionnels qui n'ont normalement pas besoin de créer des tables (même s'ils pourront le faire). Ce rôle autorise simplement d'utiliser Oracle : il permet de créer des tables, des vues, des séquences, des clusters, des synonymes, des sessions et des liens vers d'autres bases de données.

•RESOURCE : Pour les utilisateurs réguliers. Accorde des droits supplémentaires pour la création de tables, de séquences, de procédures, de déclencheurs, d'index et de clusters.

•DBA : Regroupe tous les privilèges de niveau système avec des quotas d'espace illimités et la possibilité d'accorder n'importe quel privilège à un autre utilisateur. Le compte système est employé par un utilisateur disposant d'un rôle DBA.

3. Les privilèges

GRANT <privilège_système> | <rôle> [,<privilège_système> | <rôle>, …] TO <utilisateur> | <rôle> [,<utilisateur> | <rôle>, …] [WITH ADMIN OPTION]

La commande GRANT permet d'accorder n'importe quel privilège système ou rôle à un utilisateur, à un rôle, ou au groupe d'utilisateurs PUBLIC. Si la clause WITH ADMIN OPTION est spécifiée, le bénéficiaire peut à son tour assigner le privilège ou le rôle qu'il a reçu à d'autres utilisateurs ou rôles.

Celui qui attribut un privilège à un rôle peut aussi le révoquer :

REVOKE <privilège_système> | <rôle> [,<privilège_système> | <rôle>, …] TO <utilisateur> | <rôle> [,<utilisateur> | <rôle>, …] …

Un utilisateur qui dispose du rôle de DBA peut révoquer les privilèges CONNECT, RESSOURCE, DBA ou tout autre privilège ou rôle accordés à un autre utilisateur ou administrateur de base de données. Cette commande est donc très dangereuse, c'est pourquoi les privilèges DBA doivent être accordés uniquement aux personnes dont la fonction l'exige.

Nota : Révoquer tous les privilèges d'un utilisateur ne supprime ni son compte ni les objets qu'il possède. Cela l'empêche simplement d'y accéder. Les autres utilisateurs disposant d'un accès aux objets de cet utilisateur peuvent continuer à y accéder comme si de rien n'était.

Pour supprimer l'utilisateur et tous les objets qu'il possède (CASCADE) :

DROP USER <utilisateur> [CASCADE];

Un utilisateur peut accorder des privilèges d'accès à tout objet qu'il possède, alors que l'administrateur de la base peut octroyer n'importe quel privilège système, car le rôle DBA inclut les privilèges GRANT ANY et GRANT ANY ROLE.

L'utilisateur peut accorder les privilèges suivants :

•Sur les tables, vues et snapshot qu'il possède : INSERT, UPDATE, DELETE, SELECT.

•Sur les tables qu'il possède : ALTER, REFERENCES, INDEX, ALL (tous les privilèges évoqués).
•Sur les procédures, fonctions, packages, types de données abstraits qu'il possède : EXECUTE.

•Sur les séquences qu'il possède : SELECT, ALTER.

Administration paramétrée

1. Création de rôles
On peut ajouter à la liste des rôles par défaut d'Oracle (CONNECT, RESOURCE, DBA) des rôles comprenant des privilèges de niveau système ou objet, ou une combinaison des deux.

Pour pouvoir créer des rôles il faut avoir le privilège système CREATE ROLE.

L'instruction suivante permet de créer un rôle :

CREATE ROLE <nom_rôle> [NOT IDENTIFIED | IDENTIFIED [BY <mot_de_passe | EXTERNALLY]];

Lorsqu'ils viennent d'être créés, les rôles ne sont associés à aucun privilège.

2. Activer ou désactiver des rôles

ALTER USER <utilisateur> DEFAULT ROLE [<rôle1>, <rôle2>] [ALL | ALL EXECPT <rôle1>, <rôle2>] [NONE]

ALTER ROLE <nom_rôle>

3. Pour désactiver un rôle

DROP ROLE <nom_rôle>;

Activer ou désactiver un rôle :

SET ROLE <nom_rôle>;

SET ROLE <nom_rôle> NONE;

4. Révoquer les privilèges d'un rôle

REVOKE <privilège> FROM <nom_rôle>;

Exemple : REVOKE SELECT ON PRODUIT FROM CLERK;

5. Suppression d'un rôle

DROP ROLE <nom_rôle>;

Mettre en place les déclencheurs

La syntaxe des déclencheurs sous Oracle peut être plus ou moins complexe.

```
CREATE [ OR REPLACE ] TRIGGER schéma.nom_déclencheur
AFTER | BEFORE { [ INSERT [ OR DELETE
          [ OR UPDATE OF nom_colonne,...,nom_colonneN ] ] ] }
ON schéma.nom_table
FOR EACH ROW nom_procédure (argument...argumentN);
```

La seconde syntaxe se révèle bien plus absconse, puisqu'elle intègre non seulement les déclencheurs LMD mais également d'autres types de **déclencheurs basés sur des commandes LDD ou *DATABASE***.

En outre, pour les déclencheurs LMD, **des clauses spécifiques aux vues** apparaissent, ainsi que **la clause *REFERENCING***.

```
CREATE [ OR REPLACE ] TRIGGER [ schéma. ] nom_déclencheur
{
  AFTER | BEFORE | INSTEAD OF
}
{
  {
  [ INSERT
  [ OR DELETE
  [ OR UPDATE [ OF nom_colonne [, nom_colonneN ] ] ] ] ]
  }
  ON
  {
   { [ schéma. ] nom_table }
        |
   { [ NESTED TABLE colonne_emboîtée OF ] [ schéma . ] nom_vue }
  }
  {
   [ REFERENCING ]
```

```
    {
        [ OLD [ AS ] ancienne_colonne
        | NEW [ AS ] nouvelle_colonne
        | PARENT [ AS ] colonne_parente ]
    }
  }
  [ FOR EACH ROW ] instruction_SQL;
}
  |
{
  {
  { Evénement_LDD [ OR EvénementN_LDD ] }
      |
  { Evénement_Base_Données [ OR EvénementN_Base_Données ]
  }
  ON
  {
  { [ schéma. ] SCHEMA } | DATABASE
  }
  WHEN ( Condition )
   { instruction_PL/SQL | instruction_procédure };
}
```

La commande *OR REPLACE* recrée le déclencheur s'il existe déjà.

La clause *BEFORE* indique que le déclencheur doit être lancé avant l'exécution de l'événement.

La clause *AFTER* indique que le déclencheur doit être lancé après l'exécution de l'événement.

Les instructions *INSERT* et *DELETE* indique au déclencheur de s'exécuter lors respectivement d'une insertion ou d'une suppression dans la table.

La clause *UPDATE OF* indique que le déclencheur doit être lancé lors de chaque mise à jour d'une des colonnes spécifiées. Si elle est omise, n'importe quelle colonne de la table modifiée provoque le déclenchement du *Trigger*.

La clause *ON* désigne le nom de la table associé à son schéma pour lequel le déclencheur a été spécifiquement créé.

La clause *FOR EACH ROW* désigne le déclencheur pour être un déclencheur de ligne. Oracle lance un déclencheur de ligne une fois pour chaque ligne qui est affectée par l'instruction de déclenchement. Si la clause est omise, le déclencheur est un déclencheur d'instruction. Oracle lance un déclencheur d'instruction une fois seulement lorsque l'instruction déclenchante est émise si la contrainte du déclencheur optionnelle est rencontrée.

La clause *REFERENCING* permet de spécifier des noms de corrélation. Il est possible d'utiliser les noms de corrélation dans des blocs d'instructions PL/SQL et dans la condition WHEN d'un déclencheur de ligne pour se référer spécifiquement à des valeurs anciennes et nouvelles de la ligne courante. Les noms de corrélation par défaut sont *OLD* et *NEW*. Si le déclencheur de ligne est associé à une table nommée *OLD* ou *NEW*, l'utilisation de cette clause permet de spécifier des noms de corrélations différents afin d'éviter une confusion entre les noms de table et les noms de corrélation.

Si le déclencheur est défini sur une table imbriquée, *OLD* et *NEW* se réfèrent à la ligne de la table imbriquée, et *PARENT* se réfère à la ligne courante de la table parente.
Si le déclencheur est défini sur une table ou une vue, *OLD* et *NEW* se réfère aux instances d'objet.
La clause *REFERENCING* n'est pas valide avec les déclencheurs *INSTEAD OF* sur les événements LDD de création.

Exemple

```
-- Premier exemple
CREATE OR REPLACE TRIGGER declencheur_suppression
  AFTER DELETE ON tbl_1
  FOR EACH ROW
  WHEN (1 = 1)
  DECLARE action_utilisateur VARCHAR2(50);
  BEGIN
  SELECT user INTO action_utilisateur FROM DUAL;
  INSERT INTO delete_log
  VALUES ('tbl_1',action_utilisateur,TO_CHAR(SYSDATE, 'DD/MON/YYYY'),
                          TO_CHAR(SYSDATE, 'HH24:MI:SS'));
  END;

-- Second exemple
CREATE TABLE tbl_1 (col_a INTEGER, col_b CHAR(20));
CREATE TABLE tbl_2 (col_c CHAR(20), col_d INTEGER);

CREATE TRIGGER declencheur_insertion
  AFTER INSERT ON tbl_1
  FOR EACH ROW
  WHEN (NEW.col_a <= 10)
  BEGIN
      INSERT INTO tbl_2 VALUES(:NEW.col_b, :NEW.col_a);
  END;
```

Le premier exemple crée un déclencheur qui insère un champ log à l'intérieur d'une table, pour chaque ligne supprimée dans la table spécifiée.

Le second exemple crée un déclencheur qui insère un enregistrement à l'intérieur de la seconde table lorsqu' une opération d'insertion s'est accomplie dans la première table. Le déclencheur vérifie si le nouvel enregistrement possède un premier composant inférieur ou égal à 10 et si c'est le cas, inverse les enregistrements à l'intérieur de la seconde table.

Les variables spéciales *NEW* et *OLD* sont disponibles pour se référer respectivement à des nouveaux ou d'anciens enregistrements. Les deux points (:) précédent *NEW* et *OLD* dans *VALUES* sont dans ce cas obligatoires, par contre dans la clause conditionnelle *WHEN*, ils doivent être omis.