Chapitre 4

Langage de définition des données

Le "Langage de Définition des Données" est la partie de SQL qui permet de décrire les tables et autres objets manipulés par ORACLE.

4.1 Schéma

Un schéma est un ensemble d'objets (tables, vues, index, etc...) gérées ensemble. Cette notion correspond au concept habituel de base de données. On pourra ainsi avoir un schéma lié à la gestion du personnel et un autre lié à la gestion des clients.

Cette notion introduite par la norme SQL-2 n'est pas vraiment prise en compte par Oracle qui identifie pour le moment un nom de schéma avec un nom d'utilisateur.

4.2 Tables

4.2.1 CREATE TABLE AS

La commande CREATE a déjà été vue au premier chapitre. Une variante permet d'insérer pendant la création de la table des lignes venant d'autres tables :

```
CREATE TABLE table (col type.....)
AS SELECT .....
```

Exemple 4.1

```
CREATE TABLE MINIDEPT(CLE NUMBER, DATA CHAR(20))
AS SELECT DEPT, NOMD FROM DEPT
```

Cet ordre créera une table MINIDEPT et la remplira avec deux colonnes des lignes de la table DEPT.

Il faut évidemment que les définitions des colonnes de la table créée et du résultat de la sélection soient compatibles en type et en taille.

On peut également spécifier le mot-clé AS et l'interrogation directement derrière le nom de la table. Dans ce cas les colonnes de la table créée auront les mêmes noms, types et tailles que celles de l'interrogation :

```
CREATE TABLE DEPT10 AS SELECT * FROM DEPT WHERE DEPT = 10
```

4.2.2 ALTER TABLE

Oracle offre la possibilité de modifier la définition d'une table. Deux types de modifications sont possibles : ajout d'une colonne (après toutes les autres colonnes), et modification d'une colonne existante.

Il n'est pas possible de supprimer une colonne mais les valeurs d'une colonne qui n'est plus utilisée peuvent être mises à la valeur NULL.

4.2.3 Ajout d'une colonne - ADD

```
ALTER TABLE table
ADD (col1 type1, col2 type2, ...)
```

permet d'ajouter une ou plusieurs colonnes à une table existante. Les types possibles sont les mêmes que ceux décrits avec la commande CREATE TABLE. Les parenthèses ne sont pas nécessaires si on n'ajoute qu'une seule colonne.

L'attribut 'NOT NULL' peut être spécifié seulement si la table est vide (si la table contient déjà des lignes, la nouvelle colonne sera nulle dans ces lignes existantes et donc la condition 'NOT NULL' ne pourra être satisfaite).

4.2.4 Modification d'une colonne - MODIFY

```
ALTER TABLE table
MODIFY (col1 type1, col2 type2, ...)
```

col1, *col2*... sont les noms des colonnes que l'on veut modifier. Elles doivent bien sûr déjà exister dans la table. *type1*, *type2*,... sont les nouveaux types que l'on désire attribuer aux colonnes.

Il est possible de modifier la définition d'une colonne, à condition que la colonne ne contiennent que des valeurs NULL ou que la nouvelle définition soit compatible avec le contenu de la colonne :

- on ne peut pas diminuer la taille maximale d'une colonne.
- on ne peut spécifier 'NOT NULL' que si la colonne ne contient pas de valeur nulle.

Mais il est toujours possible d'augmenter la taille maximale d'une colonne, tant qu'on ne dépasse pas les limites propres à SQL, et on peut dans tous les cas spécifier 'NULL' pour autoriser les valeurs nulles.

4.2.5 DROP TABLE

`DROP TABLE table`
permet de supprimer une table : les lignes de la table et la définition elle-même de la table sont détruites. L'espace occupé par la table est libéré.

4.3 Vues

On peut enregistrer un ordre SELECT en tant que vue. Les utilisateurs pourront consulter la base, ou modifier la base (avec certaines restrictions) à travers la vue, c'est-à-dire manipuler la table résultat du SELECT comme si c'était une table réelle.

Seule la définition de la vue est enregistrée dans la base, et pas les données de la vue. On peut parler de table virtuelle.

4.3.1 CREATE VIEW

La commande CREATE VIEW permet de créer une vue en spécifiant le SELECT constituant la définition de la vue :

`CREATE VIEW vue (col1, col2...) AS SELECT ...`

La spécification des noms des colonnes de la vue est facultative : par défaut, les colonnes de la vue ont pour nom les noms des colonnes résultat du SELECT. Si certaines colonnes résultat du SELECT sont des expressions sans nom, il faut alors obligatoirement spécifier les noms de colonnes de la vue.

Le SELECT peut contenir toutes les clauses d'un SELECT, sauf la clause ORDER BY.

Exemple 4.2

Vue constituant une restriction de la table EMP aux employés du département 10 :

```
CREATE VIEW EMP10 AS
SELECT * FROM EMP
WHERE DEPT = 10
```

Remarque 4.1

Dans l'exemple ci-dessus il aurait été plus prudent et plus souple d'éviter d'utiliser "*" et de le remplacer par les noms des colonnes de la table EMP. En effet, si la définition de la table EMP est modifiée, il y aura une erreur à l'exécution si on ne reconstruit pas la vue EMP10.

4.3.2 DROP VIEW

`DROP VIEW vue`
supprime la vue "*vue*".

4.3.3 Utilisation des vues

Une vue peut être référencée dans un SELECT de la même façon qu'une table. Ainsi, il est possible de consulter la vue EMP10. Tout se passe comme s'il existait une table EMP10 des employés du département 10 :

```
SELECT * FROM EMP10
```

Mise à jour avec une vue

Il est possible d'effectuer des INSERT et des UPDATE à travers des vues, sous deux conditions :

- le SELECT définissant la vue ne doit pas comporter de jointure,
- les colonnes résultats du SELECT doivent être des colonnes réelles et non des expressions composées.

Ainsi, il est possible de modifier les salaires du département 10 à travers la vue EMP10. Toutes les lignes de la table EMP avec DEPT = 10 seront modifiées :

```
UPDATE EMP10
SET SAL = SAL * 1.1
```

Une vue peut créer des données qu'elle ne pourra pas visualiser. On peut ainsi ajouter un employé du département 20 avec la vue EMP10.

Si l'on veut éviter cela il faut ajouter "**WITH CHECK OPTION**" dans l'ordre de création de la vue après l'interrogation définissant la vue. Il est alors interdit de créer au moyen de la vue des lignes qu'elle ne pourrait relire. Ce dispositif fonctionne également pour les mises à jour.

Exemple 4.3

```
CREATE VIEW EMP10 AS
SELECT * FROM EMP
WHERE DEPT = 10
WITH CHECK OPTION
```

4.3.4 Utilité des vues

De façon générale, les vues permettent de dissocier la façon dont les utilisateurs voient les données, du découpage en tables. On sépare l'aspect externe (ce que voit un utilisateur particulier de la base) de l'aspect conceptuel (comment a été conçu l'ensemble de la base). Ceci favorise l'indépendance entre les programmes et les données. Si la structure des données est modifiée, les programmes ne seront pas à modifier si l'on a pris la précaution d'utiliser des vues (ou si on peut se ramener à travailler sur des vues). Par exemple, si une table est découpée en plusieurs tables après l'introduction de nouvelles données, on peut introduire une vue, jointure des nouvelles tables, et la nommer du nom de l'ancienne table pour éviter de réécrire les programmes qui utilisaient l'ancienne table.

Une vue peut aussi être utilisée pour restreindre les droits d'accès à certaines colonnes et à certaines lignes d'une table : un utilisateur peut ne pas avoir accès à une table mais avoir les autorisations pour utiliser une vue qui ne contient que certaines colonnes de la table ; on peut de plus ajouter des restrictions d'utilisation sur cette vue comme on le verra en 4.5.1.

Dans le même ordre d'idées, une vue peut être utilisée pour implanter une contrainte d'intégrité grâce à l'option "WITH CHECK OPTION".

Une vue peut également simplifier la consultation de la base en enregistrant des SELECT complexes.

Exemple 4.4

En créant la vue :

```
CREATE VIEW EMPSAL AS
  SELECT NOME, SAL + NVL(COMM, 0) GAINS, NOMD
  FROM EMP, DEPT
  WHERE EMP.DEPT = DEPT.DEPT
```

La liste des employés avec leur rémunération totale et le nom de leur département sera obtenue simplement par :

```
SELECT * FROM EMPSAL
```

4.4 Index

Considérons le SELECT suivant :

```
SELECT * FROM EMP
WHERE NOME  = 'MARTIN'
```

Un moyen de retrouver la ou les lignes pour lesquelles NOME est égal à 'MARTIN' est de balayer toute la table.

Un tel moyen d'accès conduit à des temps de réponse prohibitifs pour des tables dépassant quelques milliers de lignes.

Une solution est la création d'index, qui permettra de satisfaire aux requêtes les plus fréquentes avec des temps de réponses acceptables.

Un index est formé de clés auxquelles SQL peut accéder très rapidement. Comme pour l'index d'un livre, ces clés permettent de lire ensuite directement les données repérées par les clés.

4.4.1 CREATE INDEX

Un index se crée par la commande CREATE INDEX :

```
CREATE [UNIQUE] INDEX nom-index ON table (col1, col2,...)
```


On peut spécifier par l'option "UNIQUE" que chaque valeur d'index doit être unique dans la table.

Remarque 4.2

Deux index construits sur des tables d'un même utilisateur ne peuvent avoir le même nom (même s'ils sont liés à deux tables différentes).

4.4.2 Utilisation des index

Un index peut être créé juste après la création d'une table ou sur une table contenant déjà des lignes. Il sera ensuite tenu à jour automatiquement lors des modifications de la table.

Un index peut porter sur plusieurs colonnes : la clé d'accès sera la concaténation des différentes colonnes.

On peut créer plusieurs index indépendants sur une même table.

Les requêtes SQL sont transparentes au fait qu'il existe un index ou non. C'est l'optimiseur du SGBD qui, au moment de l'exécution de chaque requête, recherche s'il peut s'aider d'un index.

La principale utilité des index est d'accélérer les recherches d'informations dans la base. Une ligne est retrouvée instantanément si la recherche peut utiliser un index. Sinon, une recherche séquentielle sur toutes les lignes de la table doit être effectuée. Il faut cependant noter que les données à retrouver doivent correspondre à environ moins de 20 % de toutes les lignes sinon une recherche séquentielle est préférable.

Les index concaténés permettent même dans certains cas de récupérer toutes les données cherchées sans accéder à la table.

Une jointure s'effectuera souvent plus rapidement si une des colonnes de jointure est indexée (s'il n'y a pas trop de valeurs égales dans les colonnes indexées).

Les index accélèrent le tri des données si le début de la clé de tri correspond à un index.

Une autre utilité des index est d'assurer l'unicité d'une clé en utilisant l'option "UNIQUE". Ainsi la création de l'index suivant empêchera l'insertion dans la table EMP d'un nom d'employé existant :

```
CREATE UNIQUE INDEX NOME ON EMP (NOME)
```

Il faut cependant savoir que les modifications des données sont ralenties si un ou plusieurs index doivent être mis à jour. De plus, la recherche d'information n'est accélérée par un index que si l'index ne contient pas trop de données égales. Il n'est pas bon, par exemple, d'indexer une colonne SEXE qui ne pourrait contenir que des valeurs "M" ou "F".

4.4.3 DROP INDEX

Un index peut être supprimé par la commande DROP INDEX :

```
DROP INDEX nom_index [ON table]
```