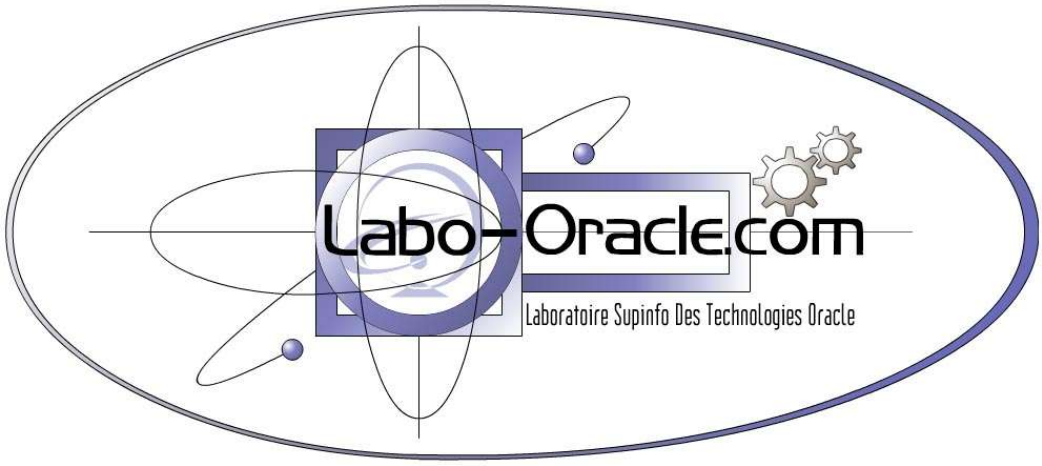


Module n°4

GESTION DES DONNEES

1Z0-031

Auteur : Mathieu Musard

GESTION DES DONNEES – d septembre yyyy

Nombre de pages : 43

Paris Academy of Computer Science
SUPINFO Paris
Ecole Supérieure d'Informatique

Ecole Supérieure d'Informatique de Paris
23. rue Château Landon 75010 – PARIS
www.supinfo.com

Table des matières

1.GÉRER UNE TABLE..... 4

1.1. Stocker les données utilisateurs..... 4
1.1.1. Utiliser différentes méthodes de stockage des données utilisateur..... 4
1.1.2.Tables normales..... 4
1.1.3.Table partitionnée..... 4
1.1.4.Tables organisées en index..... 4
1.1.5.Table incluse dans un cluster..... 4
1.2.Structure d'une ligne..... 5
1.2.1.Format et longueur de ligne..... 5
1.2.2.Types de données internes Oracle..... 6
1.2.3.Types de données scalaires..... 6
1.2.4.Types de données Ensemble..... 9
1.3. Créer une table..... 9
1.3.1.Copier une table existante..... 11
1.3.2.Tables temporaires..... 11
1.3.3.Instructions de création d'une table..... 11
1.3.4.Définir le paramètre PCTFREE..... 12
1.3.5.Définir le paramètre PCTUSED..... 12
1.3.6.Migrer des lignes..... 12
1.3.7.Chaîner des lignes..... 12
1.4. Gérer l'espace utilisé par les tables..... 12
1.4.1.Modifier les paramètres d'utilisation de bloc et de stockage..... 12
Syntaxe..... 12
1.4.2.Impact de la modification des paramètres de stockage..... 13
1.4.3.Paramètres d'utilisation de blocs..... 13
1.4.4.Allouer manuellement des extents..... 13
1.4.5.Réorganiser les tables non partitionnées..... 14
1.4.6.Tronquer une table..... 15
1.4.7.Supprimer une table..... 15
1.4.8.Supprimer une colonne d'une table..... 16
1.4.9.Utiliser l'option UNUSED..... 16
1.4.10.Extraire les informations relatives à une table..... 17
1.4.11.Extraire les informations sur les extents..... 18
1.4.12.Package DBMS_ROWID..... 18

2.GESTION D'INDEX..... 21

2.1.Types d'index..... 21
2.1.1.Classification des index..... 21
2.1.2.Index B-tree..... 22
2.1.3.Index à clé inversée..... 23
2.1.4.Index bitmap..... 23
2.1.5.Index bitmap ou index B-tree..... 23
2.2.Création d'index..... 24
2.2.1.Contexte de création d'index..... 24
2.2.2.Création d'un index B-tree..... 24
2.2.3.Création d'un index à clé inversée..... 25
2.2.4.Création d'un index bitmap..... 25
2.3.Gestion d'index..... 26
2.3.1.Modification des paramètres de stockage..... 26
2.3.2.Allocation et desallocation d'espace dans l'index..... 26
2.3.3.Libération dans l'index..... 27
2.3.4.Situations de reconstruction d'un index..... 27
2.3.5.Caractéristiques de reconstruction d'un index..... 28
2.3.6.Index coalescents..... 28
2.3.7.Contrôle de la validité d'un index..... 29
2.3.8.Suppression d'un index..... 29
2.3.9.Utilisation du monitoring pour identifier les index inutilisés..... 29

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

2.3.10.Informations sur les index..... 30

3.GÉRER L'INTÉGRITÉ DES DONNÉES.....32

3.1. INTÉGRITÉ DES DONNÉES..... 32
3.1.1.Méthodes garantissant l'intégrité des données..... 32
3.2. METTRE EN ŒUVRE DES CONTRAINTES.....37
3.2.1.Définir les contraintes lors de la création d'un table..... 37
3.2.2.Syntaxe pour les contraintes de colonne ou de table..... 37
3.2.3.Syntaxe : contrainte de table..... 38
3.2.4.Définir des contraintes après la création d'une table :..... 38
3.2.5.Instructions de définition des contraintes..... 38
3.3. GÉRER LES CONTRAINTES..... 39
3.3.1.Activer les Contraintes 39
3.3.2.Utiliser la table EXCEPTIONS..... 40
Obtenir des informations sur les contraintes..... 41

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

1.Gérer une table

1.1. Stocker les données utilisateurs

1.1.1. Utiliser différentes méthodes de stockage des données utilisateur

Dans une base de données Oracle, vous pouvez stocker les données utilisateur de différentes manières. Les données peuvent être stockées dans l'une des tables suivantes :

- tables normales,
- tables partitionnées,
- tables organisées en index,
- tables incluses dans un cluster.

1.1.2.Tables normales

Une table normale (appelée en général "table") est l'élément le plus communément utilisé pour stocker les données utilisateur. Il s'agit du type de table par défaut qui constitue le sujet principal de chapitre. Les prérogatives de l'administrateur sur la distribution des lignes d'une table incluse dans un cluster sont très limitées. Les lignes peuvent être stockées dans n'importe quel ordre selon l'activité de la table.

1.1.3.Table partitionnée

Une table partitionnée permet de créer des applications évolutives.
Elle présente les caractéristiques suivantes :

- Une table partitionnée contient une ou plusieurs partitions qui stockent chacune les lignes partitionnées à l'aide du partitionnement par plage, par hachage ou composite.
- Chaque partition d'une table partitionnée constitue un segment qui peut se trouver dans un tablespace différent.
- Les partitions trouvent une utilité particulière pour les tables volumineuses qui peuvent être interrogées ou manipulées par plusieurs processus simultanés.
- Des commandes spéciales permettent de gérer les partitions d'une table.

1.1.4.Tables organisées en index

Une table organisée en index est une table normale contenant un index de clé primaire dans une ou plusieurs de ses colonnes. Toutefois, au lieu de gérer deux espaces de stockage distincts pour la table et un index B-Tree, une table organisée en index ne gère qu'un seul B-Tree contenant la clé primaire de la table et d'autres valeurs de colonnes.
Les tables organisées en index accélèrent l'accès par clé aux données des tables dans les interrogations impliquant des recherches de correspondance exacte ou des recherches sur des plages de données.
En outre, les besoins en stockage sont moindres dans la mesure où les colonnes de clé ne sont pas dupliquées dans la table, ni dans l'index. Les colonnes restantes ne contenant pas de clé sont stockées dans l'index si la taille de l'entrée d'index n'augmente pas considérablement. Si tel n'est pas le cas, le serveur Oracle fournit la clause OVERFLOW qui permet de résoudre le problème.

1.1.5.Table incluse dans un cluster

Une table incluse dans un cluster permet également de stocker les données dans des tables. Un cluster est constitué de plusieurs tables qui partagent les mêmes blocs de données, regroupés du fait qu'ils partagent des colonnes et souvent utilisés conjointement.
Un cluster présente les caractéristiques suivantes :

- Il dispose d'une clé de cluster qui permet d'identifier les lignes devant être enregistrées ensembles.

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

- La clé de cluster peut être constituée d'une ou de plusieurs colonnes.
- Les tables d'un cluster contiennent des colonnes correspondant à la clé de cluster.
- La création de cluster est un mécanisme qui est transparent aux applications qui utilisent les tables. Les données d'une table incluse dans un cluster peuvent être manipulées de la même manière que celles d'une table normale.
- La mise à jour de l'une des colonnes de la clé de cluster peut provoquer le déplacement physique de la ligne.
- La clé de cluster est indépendante de la clé primaire. Les tables d'un cluster peuvent avoir une clé primaire qui peut correspondre à la clé de cluster ou à un ensemble de colonnes différent.
- Les clusters sont généralement utilisés pour améliorer les performances. L'accès direct aux données d'un cluster peut être plus rapide, mais le balayage complet de tables est généralement plus long.

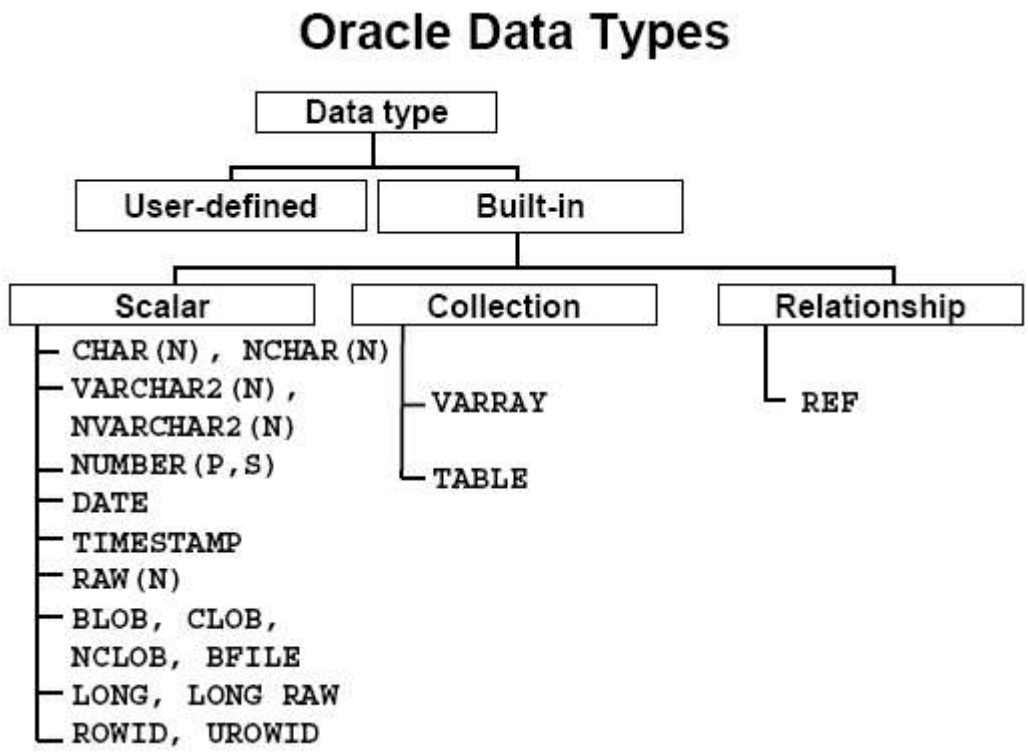
1.2.Structure d'une ligne

1.2.1.Format et longueur de ligne

Les données des lignes sont stockées dans des blocs de base de données sous forme d'enregistrements de longueur variable. Les colonnes d'une ligne sont généralement enregistrées selon l'ordre dans lequel elles sont définies, et toutes les colonnes de fin NULL ne sont pas enregistrées. Chaque ligne d'une table peut comporter un nombre de colonnes différent. Chaque ligne d'une table a :

- un en-tête : cet en-tête permet de stocker le nombre de colonnes de la ligne, les informations de chaînage et le statut de verrouillage de la ligne,
- des données de ligne : le serveur Oracle stocke la longueur et la valeur de chaque colonne (un octet est nécessaire pour enregistrer la longueur de colonne si celle-ci ne peut dépasser 250 octets. Une colonne plus longue nécessite 3 octets. La valeur de la colonne est stockée immédiatement après les octets de longueur de colonne).

Aucun espace n'est nécessaire entre les lignes adjacentes. Chaque ligne du bloc est associée à un pointeur dans le répertoire des lignes. Le pointeur pointe vers le début de la ligne.



Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

1.2.2.Types de données internes Oracle

Le serveur Oracle fournit plusieurs types de données internes pour stocker les données scalaires, les ensembles et les relations.

1.2.3.Types de données scalaires

Données de type caractère

Les données de type caractère sont stockées sous forme de chaînes de longueur fixe ou de longueur variable dans la base de données.
Les données de longueur fixe de type caractère, tel que CHAR et NCHAR, sont stockées avec des espaces. NCHAR est un type de données NLS (National Language-Supported) qui permet de stocker des jeux de caractères de longueur fixe ou de longueur variable. La taille maximale est déterminée par le nombre d'octets nécessaires au stockage d'un caractère, avec une limite maximale de 2 000 octets par ligne. La valeur par défaut est de 1 caractère, soit 1 octet, selon le jeu de caractères.
Les données de longueur variable de type caractère utilisent uniquement le nombre d'octets nécessaires au stockage de la valeur de colonne réelle et leur taille peut varier pour chaque ligne et être égale à 4 000 octets au maximum. Les données de type VARCHAR2 et NVARCHAR2, par exemple, sont des données de type caractère de longueur variable.

Données numériques

Dans les bases de données Oracle, les nombres sont toujours stockés sous forme de données de longueur variable. Ils peuvent comporter jusqu'à 38 chiffres significatifs. Les données numériques nécessitent :

- un octet pour l'exposant,
- un octet tous les deux chiffres significatifs dans la mantisse,
- un octet pour les nombres négatifs si le nombre de chiffres significatifs est inférieur à 38 octets.

Données de type DATE

Le serveur Oracle enregistre les dates dans des champs de longueur fixe de sept octets. Une date Oracle contient toujours l'heure.

Données de type TIMESTAMP

Ce type de données stocke la date et le temps avec une précision de 9 chiffres après la virgule. TIMESTAMP WITH TIME ZONE et TIMESTAMP WITH LOCAL TIME ZONE peuvent être utilisés pour gérer l'heure d'été et d'hiver. TIMESTAMP et TIMESTAMP WITH LOCAL TIME ZONE peuvent être utilisés en clés primaires contrairement à TIMESTAMP WITH TIME ZONE.

Données de type RAW

Ce type de données permet de stocker de petites données binaires. Le serveur Oracle ne convertit pas le jeu de caractères lorsque des données de type RAW sont transmises entre les machines d'un réseau ou lorsqu'elles sont transféré es d'une base de données vers une autre à l'aide d'utilitaires Oracle. Le nombre d'octets nécessaires au stockage de la valeur de colonne réelle varie pour chaque ligne et peut être é gal à 2 000 octets au maximum.

Types de données scalaires de stockage des objets LOB

Oracle fournit six types de données pour stocker les objets LOB :

- CLOB et LONG pour les grands caractères de longueur fixe,
- NCLOB pour les jeux de grands caractères nationaux de longueur fixe,
- BLOB et LONG RAW pour stocker les données non structurées,

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

- BFILE pour stocker les données non structurées dans des fichiers de système d'exploitation.

Les types de données LONG et LONG RAW étaient utilisés auparavant pour les données non structurées, telles que les images binaires, les documents ou les informations géographiques, et sont principalement fournis à des fins de compatibilité descendante. Ces types de données sont remplacés par les types de données LOB. Les données de type LOB sont différentes des données de type LONG et LONG RAW et ne sont pas interchangeables. Les données de type LOB ne prennent pas en charge l'interface API LONG (Application Programming Interface), et vice versa.

Comparer les données de type LONG et LOB

La comparaison entre la fonctionnalité LOB et les anciens types de données peut s'avérer utile. Ci-dessous, LONG fait référence à LONG et LONG RAW, et LOB à toutes les données de type LOB.
Les données de type LOB permettent d'utiliser plusieurs colonnes LOB par table ou plusieurs attributs dans un type d'objet, alors que le type LONG n'en accepte qu'une seule ou qu'un seul.
La taille maximale des données de type LONG est de 2 gigaoctets alors que celle des données de type LOB peut être égale à 4 gigaoctets maximum.
Lors de leur extraction, les types de données LOB renvoient le pointeur alors que les types de données LONG renvoient les données.
Les types de données LOB stockent un pointeur dans la table et les données dans un autre emplacement, sauf si la taille des données de type VARCHAR2 est inférieure à la taille maximale de 4 000 octets. Les types de données LONG stockent toutes les données dans le segment. En outre, les types de données LOB permettent de stocker les données dans un segment et un tablespace distincts ou dans un fichier hôte.
Les données de type LOB prennent en charge les attributs de type d'objet (sauf le type de données NCLOB), contrairement aux données de type LONG.
Les données de type LONG sont stockées sous forme de morceaux de lignes chaînés, un morceau de ligne d'un bloc pointant vers le morceau de ligne suivant d'un autre bloc. En conséquence, l'accès aux segments s'effectue séquentiellement. En revanche, les données de type LOB permettent d'accéder directement aux segments de données via une interface de type fichier.

Type de données ROWID et UROWID

ROWID est une pseudo colonne qui peut être interrogée en même temps que les autres colonnes d'une table. Ce type de données présente les caractéristiques suivantes :

- Il identifie de manière unique chaque ligne dans une base de données.
- Il n'est pas enregistré explicitement sous la forme d'une valeur de colonne.
- Bien que le type ROWID ne fournisse pas directement l'adresse physique d'une ligne, il permet de rechercher la ligne.
- Il constitue la méthode la plus rapide pour accéder à une ligne d'une table.
- Les données de type ROWID sont stockées dans des index définissant des lignes ayant des valeurs de clé spécifiques.

La version 8.1 du serveur Oracle fournit un nouveau type de données unique appelé "universale rowid" ou UROWID. Il prend en charge les données de type ROWID des tables étrangères (tables non Oracle) et peut stocker n'importe quelle donnée de ce type. Pour pouvoir utiliser le type de données UROWID, vous devez affecter la valeur 8.1 ou supérieure au paramètre COMPATIBLE.

Format des données de type ROWID

Les données de type ROWID nécessitent 10 octets de stockage sur disque et s'affichent sur 18 caractères. Ces données sont constituées des éléments suivants :

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

- *numéro d'objet de données* : affecté à chaque objet de données créé, tel qu'une table ou un index, et unique dans la base de données,
- *numéro de fichier relatif* : unique pour chaque fichier d'un tablespace,
- *numéro de bloc* : indique l'emplacement du bloc contenant la ligne dans le fichier,
- *numéro de ligne* : indique la position du pointeur du répertoire des lignes dans l'entête de bloc.

En interne, le numéro d'objet de données nécessite 32 bits, le numéro de fichier relatif, 10 bits, le numéro de bloc, 22 bits, et le numéro de ligne, 16 bits, soit un total de 80 bits, ou 10 octets.
Les données de type ROWID s'affichent en utilisant un schéma d'encodage en base 64 qui utilise six positions pour le numéro d'objet, trois positions pour le numéro de fichier relatif, six positions pour le numéro de bloc et trois positions pour le numéro de ligne. Ce schéma utilise les caractères "A-Z", "a-z", "0-9", "+" et "/"—, soit un total de 64 caractères, comme dans l'exemple ci-dessous :+

```
SQL> SELECT id, ROWID FROM summit.department;
ID ROWID
--------- ------------------
10 AAADC4AACAAAAMAAAA
31 AAADC4AACAAAAMAAAB
32 AAADC4AACAAAAMAAAC
33 AAADC4AACAAAAMAAAD
34 AAADC4AACAAAAMAAAE
35 AAADC4AACAAAAMAAAF
41 AAADC4AACAAAAMAAAG
42 AAADC4AACAAAAMAAAH
43 AAADC4AACAAAAMAAAI
44 AAADC4AACAAAAMAAAJ
45 AAADC4AACAAAAMAAAK
50 AAADC4AACAAAAMAAAL
```

Dans cet exemple :
• AAADC4 correspond au numéro d'objet de données.
• AAC représente le numéro de fichier relatif.
• AAAAMA indique le numéro de bloc.
• AAA dé finit le numéro de ligne du département dont l'ID=10.

Rechercher une ligne à l'aide du type de données ROWID

Etant donné qu'un segment ne peut résider que dans un seul tablespace, le serveur Oracle peut identifier le tablespace contenant la ligne en utilisant le numéro d'objet de données.
Le numéro de fichier relatif figurant dans le tablespace identifie le fichier, le numéro de bloc identifie le bloc contenant la ligne, et le numéro de ligne identifie la ligne dans le répertoire des lignes.
L'entrée du répertoire des lignes peut permettre de rechercher le début de la ligne. Ainsi, le type de données ROWID permet de rechercher n'importe quelle ligne dans une base de données.

Utiliser un ROWID restreint dans Oracle7 et les versions antérieures

Les versions antérieures du serveur Oracle8 utilisaient le format ROWID restreint. Un ROWID restreint utilisait seulement six octets en interne et ne contenait pas de numéro d'objet de données. Ce format était acceptable dans Oracle7 ou les versions antérieures, car les numéros de fichier étaient uniques dans une base de données.
Ainsi, les versions précédentes étaient limitées à 1 022 fichiers de données. Bien qu'Oracle8 ait supprimé cette limitation en utilisant des numéros de fichier relatifs à des tablespaces, un ROWID

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

restreint est toujours utilisé dans des objets, tels que les index non partitionnés ou les tables non partitionnées, dans lesquels toutes les entrées d'index font référence aux lignes d'un même segment.

1.2.4.Types de données Ensemble

Deux types de données Ensemble permettent de stocker les données répétitives d'une ligne d'une table. Dans les versions antérieures du serveur Oracle8i, l'option Objects était nécessaire pour définir et utiliser les ensembles.

Tableaux de taille variable (VARRAY)

Ces tableaux permettent de stocker des listes contenant un petit nombre d'éléments, tels que des numéros de téléphone de clients.
Les tableaux VARRAY présentent les caractéristiques suivantes :

- Un tableau contient des données ordonnées.
- Les données d'un tableau ont toutes le même type.
- Chaque élément comporte un index qui correspond à un numéro indiquant la position de l'élément dans le tableau.
- Le nombre d'éléments d'un tableau correspond à la taille du tableau.
- Le serveur Oracle permet d'utiliser des tableaux de taille variable, et c'est la raison pour laquelle ils s'appellent VARRAY, mais la taille maximale du tableau doit être définie lors de sa déclaration.

Tables imbriquées

Les tables imbriquées permettent de définir une table sous forme de colonne dans une table. Ces tables peuvent être utilisées pour stocker des ensembles pouvant contenir un grand nombre d'enregistrements, tels que le nombre d'articles d'une commande.
Généralement, les tables imbriquées présentent les caractéristiques suivantes :
• Une table imbriquée est un ensemble non ordonné d'enregistrements ou de lignes.
• Les lignes d'une table imbriquée présentent la même structure.
• Les lignes d'une table imbriquée sont stockées dans une table distincte de la table mère, avec un pointeur vers la ligne correspondante dans la table mère.
• Les caractéristiques de stockage d'une table imbriquée peuvent être définies par l'administrateur de base de données.
• Il n'existe pas de taille maximale prédéfinie pour une table imbriquée.

Types de données Relation (REF)

Les types de données Relation servent de pointeurs dans la base de données. L'utilisation de ces types de données fait appel à l'option Objects. Par exemple, chaque élément ordonné peut pointer vers une ligne de la table PRODUCTS ou y faire référence sans avoir à stocker la référence du produit.

Types de données définis par l'utilisateur

Le serveur Oracle permet à l'utilisateur de définir des types de données abstraits et de les utiliser dans l'application. L'utilisation de ces types de données fait appel à l'option Objects.

1.3. Créer une table

Utilisez la commande suivante pour créer une table :

```
CREATE TABLE [schema.] table
(column datatype [ , column datatype ] ...)
[TABLESPACE tablespace ]
[ PCTFREE integer ]
[ PCTUSED integer ]
[ INITRANS integer ]
[ MAXTRANS integer ]
```


Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

```
[ STORAGE storage-clause ]
[LOGGING | NOLOGGING]
[CACHE | NOCACHE] ]
```

Exemple :

```
CREATE TABLE employee(
     id NUMBER(7),
     last_name VARCHAR2(25),
     dept_id NUMBER(7))
PCTFREE 20 PCTUSED 50
STORAGE(INITIAL 200K NEXT 200K
PCTINCREASE 0 MAXEXTENTS 50)
TABLESPACE data;
```

où :	schema	est le propriétaire de la table,
	table	est le nom de la table,
	column	est le nom de la colonne,
	data type	est le type de données de la colonne,
	TABLESPACE	indique le tablespace de destination de la table à créer,
	PCTFREE	correspond à l'espace réservé dans chaque bloc (sous la forme d'un pourcentage de l'espace total moins l'en-tête de bloc) à l'augmentation de la longueur des lignes,
	PCTUSED	définit la limite inférieure de l'espace utilisé dans un bloc (après utilisation de l'espace défini par le paramètre PCTFREE) à partir de laquelle d'autres lignes peuvent être insérées,
	INITRANS	définit le nombre d'entrées de transaction pré allouées dans chaque bloc (la valeur par défaut est 1),
	MAXTRANS	limite le nombre d'entrées de transaction pouvant être allouées à chaque bloc (la valeur par défaut est 255),
	STORAGE	identifie la clause de stockage qui détermine l'allocation des extents (ensemble de blocs contigus) à la table,
	LOGGING	indique que la création de la table sera consignée dans le fichier de journalisation (redo log), (indique également que toutes les opérations suivantes effectuées sur la table sont enregistrées; il s'agit de la valeur par défaut),
	NOLOGGING	indique que la création de la table et de certains types de chargement de données ne sera pas consignée dans le fichier de journalisation,
	CACHE	indique que les blocs extraits pour cette table sont placés à l'extrémité la plus récemment utilisée de la liste LRU du cache de tampons (buffer cache), même lorsque la table est balayée en totalité,
	NOCACHE	indique que les blocs extraits de cette table sont placés à la fin de la liste LRU du cache des tampons, lorsque la table est balayée en totalité.

Remarque :

- En général, les tables doivent être créées avec une clé primaire.
- Si vous avez défini le paramètre MINIMUM EXTENT du tablespace, la taille des extents de la table est arrondie au multiple suivant le plus proche.
- Si vous omettez la clause [NO]LOGGING, l'attribut de journalisation de la table correspond par défaut à celui du tablespace dans lequel elle réside.
- Si vous affectez au paramètre MINEXTENTS une valeur supérieure à 1 et que le tablespace contient plusieurs fichiers de données, les extents sont répartis dans les différents fichiers du tablespace.

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

1.3.1. Copier une table existante

Utilisez la commande CREATE TABLE avec une sous requête pour copier une table en totalité ou partiellement.
La syntaxe simplifiée est :

```
CREATE TABLE [schema.]table
[ LOGGING | NOLOGGING ]
...
AS
subquery
```

Les autres clauses, telles que TABLESPACE, STORAGE et les paramètres d'utilisation des blocs peuvent être définis lors de la création d'une table à partir d'une autre table. Utilisez la clause NOLOGGING pour interdire la génération d'entrées de journalisation (redo log entries) et accélérer la création de la table.
Lorsque vous créez une table en copiant une table, les contraintes, les déclencheurs et les privilèges de table ne sont pas copiées. Si la table d'origine contient une colonne NOT NULL, cette colonne existe également dans la nouvelle table.

1.3.2. Tables temporaires

Outre les tables permanentes, vous pouvez créer des tables temporaires qui contiennent les données privées des sessions qui n'existent que le temps d'une transaction ou d'une session.
La commande CREATE GLOBAL TEMPORARY TABLE permet de créer une table temporaire spécifique à une transaction ou à une session. Les données d'une table temporaire de transaction et celles d'une table temporaire de session existent respectivement pendant la durée de la transaction et pendant la durée de la session.
Les données d'une session sont privées. Chaque session ne peut voir et modifier que ses propres données. Aucun verrou DML n'est placé sur les données d'une table temporaire. Les clauses qui contrôlent la durée de stockage des lignes sont :

- ON COMMIT DELETE ROWS : indique que les lignes sont uniquement visibles dans la transaction,
- ON COMMIT PRESERVE ROWS : indique que les lignes sont visibles pour toute la session.

Vous pouvez créer des index, des vues et des déclencheurs dans les tables temporaires et utiliser également les utilitaires Export et Import pour exporter et importer la définition d'une table temporaire. Toutefois, aucune donnée n'est exportée, même si vous utilisez l'option ROWS. La définition d'une table temporaire est visible dans toutes les sessions.

Exemple :

```
CREATE GLOBAL TEMPORARY TABLE employee_temp
AS SELECT * FROM employee;
```

1.3.3. Instructions de création d'une table

- Placez les tables dans des tablespaces distincts et non dans des tablespaces contenant des segments de undo, des segments temporaires et des index.
- Placez les tables dans des tablespaces gérés localement pour éviter la fragmentation.
- Pour améliorer les performances du balayage complet de tables, alignez les tailles d'extent à l'aide du paramètre d'initialisation
- Utilisez la clause CACHE pour les petites tables de référence pouvant être fréquemment utilisées.

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

1.3.4.Définir le paramètre PCTFREE

$$\frac{\text{(Taille de ligne moyenne - taille de ligne initiale) * 100}}{\text{Taille de ligne moyenne}}$$

Plus la valeur du paramètre PCTFREE est élevée, plus l'espace disponible pour la mise à jour d'un bloc d'une base de données est élevé. Définissez une valeur élevée lorsque la table contient :

- des colonnes initialement NULL, puis mises à jour à l'aide d'une valeur,
- des colonnes dont la taille risque d'augmenter à la suite d'une mise à jour.

En affectant une valeur plus élevée au paramètre PCTFREE, vous réduisez la densité de bloc, c'est-à-dire que chaque bloc contiendra moins de lignes.
La formule définie ci-dessus permet au bloc de disposer d'un espace suffisant permettant à la taille de la ligne d'augmenter.

1.3.5.Définir le paramètre PCTUSED

$$100 - \text{PCTFREE} - \frac{\text{Taille de ligne moyenne * 100}}{\text{Espace de données disponible}}$$

Définissez le paramètre PCTUSED pour permettre de renvoyer le bloc dans la liste d'espaces libres uniquement lorsqu'il existe un espace suffisant pour accepter une ligne moyenne. Si un bloc de la liste des espaces libres ne dispose pas de l'espace suffisant pour insérer une ligne, le serveur Oracle consulte le bloc suivant de la liste. Ce balayage linéaire continue jusqu'à ce qu'un bloc disposant de l'espace suffisant soit trouvé ou que la fin de la liste soit atteinte. Cette formule permet de réduire le temps de balayage de la liste d'espaces libres en augmentant la probabilité de trouver un bloc ayant l'espace libre nécessaire.

1.3.6.Migrer des lignes

Si vous affectez une valeur faible au paramètre PCTFREE, le bloc risque de ne pas contenir l'espace suffisant permettant à la longueur de la ligne d'augmenter à la suite d'une mise à jour. Dans ce cas, le serveur Oracle transfère la totalité de la ligne vers un nouveau bloc et conserve le pointeur du bloc d'origine vers le nouveau bloc. Cette opération s'appelle une migration de ligne. Lors de la migration d'une ligne, les performances des E/S sur la ligne diminuent, car le serveur Oracle doit balayer deux blocs de données pour extraire la ligne.

1.3.7.Chaîner des lignes

Un chaînage de ligne se produit lorsqu'une ligne ne peut pas tenir dans un bloc. Cette situation peut se produire lorsque la ligne contient des colonnes de très grande taille.
Dans ce cas, le serveur Oracle divise la ligne en morceaux de ligne plus petits. Chaque morceau de ligne est stocké dans un bloc avec les pointeurs associés pour extraire et assembler la totalité de la ligne. Le chaînage de lignes peut être limité en définissant une plus grande taille de bloc ou en divisant, si possible, la table en plusieurs tables contenant moins de colonnes.

1.4. Gérer l'espace utilisé par les tables

1.4.1.Modifier les paramètres d'utilisation de bloc et de stockage

Vous pouvez modifier certains paramètres de stockage et tous les paramètres d'utilisation des blocs à l'aide de la commande ALTER TABLE.

Syntaxe

```
ALTER TABLE [schema.]table
{[ storage-clause ]
```


Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

```
 [ PCTFREE integer ]
 [ PCTUSED integer ]
 [ INITRANS integer ]
 [ MAXTRANS integer]}
```

Exemple :

```
ALTER TABLE summit.employee
PCTFREE 30
PCTUSED 50
STORAGE (NEXT 500K
MINEXTENTS 2
MAXEXTENTS 100);
```

1.4.2.Impact de la modification des paramètres de stockage

Les paramètres que vous pouvez modifier et l'effet de leur modification sont les suivants :

- NEXT : lorsque le serveur Oracle alloue un autre extent à la table, la nouvelle valeur est utilisée. La taille des extents suivants augmente de la valeur définie par le paramètre PCTINCREASE.
- PCTINCREASE : la modification de la valeur du paramètre PCTINCREASE est enregistrée dans le dictionnaire de données. La nouvelle valeur est utilisée pour recalculer la valeur du paramètre NEXT lorsque l'extent suivant est alloué par le serveur Oracle. Supposons qu'une table à deux extents comporte les paramètres NEXT=10K et PCTINCREASE=0. Si vous affectez la valeur 100 au paramètre PCTINCREASE, le troisième extent alloué sera de 10 ko, le quatrième de 20 ko, et ainsi de suite.
- MINEXTENTS : vous pouvez affecter au paramètre MINEXTENTS une valeur inférieure ou égale au nombre actuel d'extents de la table. La nouvelle valeur n'entre pas immédiatement en vigueur, mais elle est prise en compte si la table est tronquée.
- MAXEXTENTS : vous pouvez affecter au paramètre MAXEXTENTS une valeur égale ou supérieure au nombre actuel d'extents de la table.

Restrictions Vous ne pouvez pas modifier la valeur du paramètre INITIAL d'une table.
La valeur du paramètre NEXT est arrondie à une valeur multiple de la taille du bloc, égale ou supérieure à la valeur définie.

1.4.3.Paramètres d'utilisation de blocs

Vous pouvez modifier la valeur des paramètres d'utilisation de blocs pour :

- améliorer l'utilisation de l'espace,
- réduire les possibilités de migration.

La modification des paramètres d'utilisation de blocs a les effets suivants :

- PCTFREE : la modification du paramètre PCTFREE affecte les insertions suivantes. Les blocs déjà remplis et donc inutilisés pour les insertions (100/PCTFREE) ne sont pas affectés jusqu'à ce qu'ils soient renvoyés dans la liste d'espaces libres. Ils ne peuvent être placés dans la liste d'espaces libres que si leur taux d'utilisation tombe sous la valeur du paramètre PCTUSED.
- PCTUSED : la modification du paramètre PCTUSED affecte tous les blocs de la table. Si une ligne est mise à jour ou supprimée, l'utilisation du bloc contenant la ligne est vérifiée et le bloc est réutilisé pour y insérer des données si son taux d'utilisation est inférieur à la valeur du paramètre PCTUSED.
- INITRANS : la modification du paramètre INITRANS n'affecte que les nouveaux blocs.
- MAXTRANS : la modification du paramètre MAXTRANS affecte tous les blocs de la table.

1.4.4.Allouer manuellement des extents

Vous pouvez être amené à allouer des extents manuellement :

- pour contrôler la distribution des extents d'une table dans les fichiers,

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

- avant de charger les données en masse pour empêcher l'augmentation dynamique de la taille des tables.

Syntaxe

Utilisez la commande suivante pour allouer un extent à une table :

```
ALTER TABLE [schema.]table
ALLOCATE EXTENT [ ([SIZE integer [K|M]]
[ DATAFILE 'filename' ]) ]
```

Si vous ne définissez pas le paramètre SIZE, le serveur Oracle utilise la taille d'extent NEXT_EXTENT de DBA_TABLES pour allouer l'extent.
Le fichier défini dans la clause DATAFILE doit appartenir au tablespace contenant la table. Sinon, l'instruction génère une erreur. Si vous n'utilisez pas la clause DATAFILE, le serveur Oracle alloue l'extent dans l'un des fichiers du tablespace contenant la table.
Remarque : la valeur NEXT_EXTENT de DBA_TABLES n'est pas affectée par l'allocation manuelle d'extents. Le serveur Oracle ne recalcule pas la taille de l'extent suivant lorsque cette commande est exécutée.
Exemples :

```
ALTER TABLE summit.employee
ALLOCATE EXTENT(SIZE 500K
DATAFILE '/DISK3/DATA01.DBF');
```

1.4.5.Réorganiser les tables non partitionnées

```
ALTER TABLE employee
MOVE TABLESPACE data1;
```

- Transfère les données dans un nouveau segment, tout en conservant les index, les contraintes, les privilèges, etc. dans la table.
- Utilisé pour transférer une table vers un tablespace différent ou pour réorganiser les extents.

Transférer ou réorganiser une table

Oracle9*i* permet de déplacer une table non partitionnée sans recourir à l'utilitaire Export ou Import. Cela s'avère utile pour :

- transférer une table d'un tablespace vers un autre,
- réorganiser la table pour supprimer la migration de lignes.

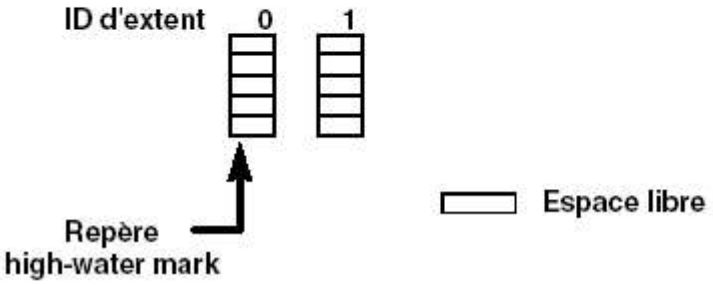
Après avoir transféré une table, vous devez reconstituer les index pour éviter l'erreur suivante :

```
SQL> select * from employee where id=23;
select * from employee where id=23
*
ERROR at line 1:
ORA-01502: index 'SUMMIT.EMPLOYEE_ID_PK' or partition of such
index is in unusable state
```


Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

1.4.6.Tronquer une table

```
TRUNCATE TABLE summit.employee;
```



La troncature d'une table supprime toutes les lignes d'une table et libère l'espace inutilisé. Dans l'exemple de la diapositive, la valeur du paramètre MINEXTENTS de la table est égale à 2.

Syntaxe

```
TRUNCATE TABLE [schema.] table
[{DROP | REUSE} STORAGE]
```

Cette commande a les effets suivants :

- Elle supprime toutes les lignes de la table.
- Aucune donnée d'annulation n'est générée et la commande valide implicitement les données, car la commande TRUNCATE TABLE est une commande LDD.
- Elle tronque les index correspondants.
- Une table référencée par une clé étrangère ne peut être tronquée.
- Les déclencheurs de suppression ne sont pas activés.
- Si vous utilisez la clause DROP (clause par dé faut) :
  - – tous les extents, à l'exception de ceux définis par le paramètre MINEXTENTS, sont libérés,
  - – le paramètre NEXT_EXTENT de la table a pour valeur la taille de l'extent libéré ayant l'ID le plus bas. Par exemple, si la valeur du paramètre MINEXTENTS est égale à 2, la valeur du paramètre NEXT_EXTENT est égale à la taille du troisième extent de la table.
- Vous devez définir la clause REUSE pour conserver tout l'espace inutilisé de la table.
- Les index sont affectés par la clause REUSE ou DROP.

1.4.7.Supprimer une table

```
DROP TABLE summit.department
CASCADE CONSTRAINTS;
```

Vous pouvez supprimer une table devenue inutile ou une table que vous souhaitez réorganiser.

Syntaxe

Utilisez la commande suivante pour supprimer une table :

```
DROP TABLE [schema.] table
[CASCADE CONSTRAINTS]
```

Lorsque vous supprimez une table, vous libérez les extents qu'elle utilise. Si les extents sont contigus, vous pouvez les fusionner automatiquement ou manuellement plus tard.

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

Remarque : vous devez utiliser l'option CASCADE CONSTRAINTS s'il s'agit de la table mère d'une relation de clé étrangère. Cette option est décrite en détail dans le chapitre "Gérer l'intégrité des données".

1.4.8.Supprimer une colonne d'une table

```
ALTER TABLE employee
DROP COLUMN comments
CASCADE CONSTRAINTS CHECKPOINT 1000;
```

- Cette commande supprime la longueur et les données de colonne de chaque ligne pour libérer de l'espace dans le bloc de données.
- La suppression d'une colonne d'une table volumineuse prend un temps considérable.

La suppression de colonnes permet de nettoyer les colonnes inutilisées et les colonnes ayant besoin d'espace, sans avoir à exporter ou à importer des données ni à recréer les index et les contraintes.
La suppression d'une colonne peut prendre un certain temps, car toutes les données de la colonne sont supprimées de la table.

Utiliser un point de reprise lors de la suppression d'une colonne

La suppression d'une colonne peut prendre un certain temps et nécessiter un espace d'annulation conséquent. Lorsque vous supprimez des colonnes dans des tables volumineuses, vous pouvez dé finir des points de reprise pour réduire l'espace d'annulation utilisé. Dans l'exemple suivant, un point de reprise se constitue toutes les 1 000 lignes. La table a l'attribut INVALID tant que l'annulation est en cours.
Si l'instance connaît un incident pendant l'annulation, la table conserve l'attribut INVALID au démarrage, et l'opération doit être poursuivie.
Utilisez l'instruction suivante pour reprendre une annulation interrompue :

```
SQL> ALTER TABLE orders
DROP COLUMNS CONTINUE;
```

Cette commande génère une erreur lorsque la table a l'attribut VALID.

1.4.9.Utiliser l'option UNUSED

- Marquer une colonne comme n'étant pas utilisée

```
ALTER TABLE orders
SET UNUSED COLUMN comments
CASCADE CONSTRAINTS;
```

- Supprimer les colonnes inutilisées

```
ALTER TABLE orders
DROP UNUSED COLUMNS CHECKPOINT 1000;
```

- Poursuivre l'opération de suppression de colonne

```
ALTER TABLE orders
DROP COLUMNS CONTINUE CHECKPOINT 1000;
```

Au lieu de supprimer une colonne d'une table, vous pouvez marquer la colonne pour indiquer qu'elle n'est pas utilisée, puis la supprimer ultérieurement. Ce marquage a l'avantage d'être relativement rapide dans la mesure où aucun espace disque n'est récupéré du fait qu'aucune donnée n'est supprimée. Vous pouvez supprimer ultérieurement les colonnes inutilisées de la table lorsque l'activité du système est faible.
Les colonnes inutilisées sont considérées comme n'appartenant pas à la table. Les interrogations ne peuvent pas voir les données des colonnes inutilisées. En outre, les noms et les types de données de ces colonnes n'apparaissent pas lorsque la commande DESCRIBE est exécutée. L'utilisateur peut ajouter une colonne en utilisant le nom d'une colonne inutilisée.

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

Vous pouvez, par exemple, marquer des colonnes comme étant inutilisées pour supprimer deux colonnes d'une même table. Lorsque vous supprimez deux colonnes d'une table, toutes les lignes de la table sont mises à jour deux fois. Si vous marquez des colonnes comme étant inutilisées et les supprimez, les lignes ne sont mises à jour qu'une seule fois.

Identifier les tables contenant des colonnes inutilisées

Pour identifier les tables contenant des colonnes inutilisées, interrogez la vue DBA_UNUSED_COL_TABS. Cette vue obtient le nom des tables contenant des colonnes inutilisées et le nombre de colonnes inutilisées qu'elles contiennent.
L'interrogation suivante indique que la table ORDERS appartenant à l'utilisateur SUMMIT contient une colonne inutilisée :

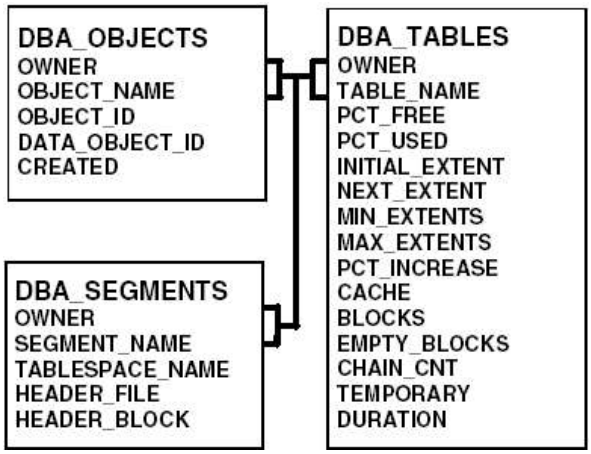
```
SQL> select * from dba_unused_col_tabs;
OWNER                          TABLE_NAME                     COUNT
------------------------------ ------------------------------ ---------
SUMMIT                         ORDERS                         1
```

Restrictions relatives à la suppression d'une colonne

Vous ne pouvez pas :

- supprimer une colonne d'une table de type objet,
- supprimer des colonnes dans des tables imbriquées,
- supprimer toutes les colonnes d'une table,
- supprimer une colonne de clé de partitionnement,
- supprimer une colonne des tables dont SYS est propriétaire,
- supprimer une colonne de clé parent,
- supprimer une colonne d'une table organisée en index si la colonne est une clé primaire.

1.4.10.Extraire les informations relatives à une table



Obtenir des informations du dictionnaire de données

Vous pouvez obtenir des informations sur les tables à partir du dictionnaire de données. Pour obtenir le numéro d'objet de données et l'emplacement de l'en-tête de table de toutes les tables appartenant à l'utilisateur SUMMIT, lancez l'interrogation suivante :

```
SQL> SELECT t.table_name, o.data_object_id,
2 s.header_file, s.header_block
3 FROM dba_tables t, dba_objects o, dba_segments s
4 WHERE t.owner=o.owner
5 AND t.table_name=o.object_name
6 AND t.owner=s.owner
7 AND t.table_name=s.segment_name
8 AND t.owner='SUMMIT';
```


Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

Obtenir des informations du dictionnaire de données (suite)

```
TABLE_NAME        DATA_OBJECT_ID      HEADER_FILE  HEADER_BLOCK
----------        --------------      -----------  ------------
CUSTOMER    12743                     2            902
DEPARTMENT        12745                     2            912
EMPLOYEE    12748                     2            927
IMAGE       12751                     2            942
INVENTORY   12753                     2            952
ITEM        12755                     2            962
LONG_TEXT   12758                     2            977
ORDERS      12760                     2            987
PRODUCT     12762                     2            997
REGION      12765                     2            1012
TITLE       12768                     2            1027
WAREHOUSE   12770                     2            1037
12 rows selected.
```

1.4.11.Extraire les informations sur les extents

DBA_EXTENTS
- **OWNER**
- **SEGMENT_NAME**
- **EXTENT_ID**
- **FILE_ID**
- **BLOCK_ID**
- **BLOCKS, TABLESPACE_NAME**

Distribuer l'espace alloué

Vous pouvez obtenir le nombre d'extents, leur emplacement et leur taille en interrogeant la vue DBA_EXTENTS. L'exemple ci-dessous indique le nombre d'extents et le nombre total de blocs utilisés par une table dans chacun des fichiers de la base de données :

```
SQL> SELECT file_id, COUNT(*) AS Extents, SUM(blocks) AS Blocks
2 FROM dba_extents
3 WHERE owner='SUMMIT'
4 AND segment_name='EMPLOYEE'
5 GROUP BY file_id;
FILE_ID     EXTENTS       BLOCKS
-------     -------       ------
3           1              25
1 row selected.
```

1.4.12.Package DBMS_ROWID

Fonctions couramment utilisées :

Nom de fonction	Description
ROWID_CREATE	Crée un identificateur ROWID à partir de composants.
ROWID_OBJECT	Renvoie l'identificateur d'objet d'un ROWID.
ROWID_RELATIVE_FNO	Renvoie le numéro de fichier relatif d'un ROWID.
ROWID_BLOCK_NUMBER	Renvoie le numéro de bloc d'un ROWID.
ROWID_ROW_NUMBER	Renvoie le numéro de ligne d'un ROWID.
ROWID_TO_ABSOLUTE_FNO	Renvoie le numéro de fichier absolu d'un ROWID.
ROWID_TO_EXTENDED	Convertit un ROWID restreint en ROWID étendu.
ROWID_TO_RESTRICTED	Convertit un ROWID étendu en ROWID restreint.

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

Obtenir des informations sur le type de données ROWID

Le serveur Oracle fournit le package DBMS_ROWID, créé à partir du script dbmsutil.sql, lui-même appelé par le script catproc.sql.
Le package contient de nombreuses fonctions permettant d'effectuer des conversions de formats ROWID et de convertir le ROWID en composants individuels. La présente section contient des exemples d'utilisation de ce package.

Obtenir les composants ROWID

Lancez l'interrogation suivante pour obtenir l'emplacement physique des lignes dans une table :

```
SQL> SELECT id, ROWID,
2 DBMS_ROWID.ROWID_OBJECT(ROWID) AS OBJECT,
3 DBMS_ROWID.ROWID_RELATIVE_FNO(ROWID) AS "RELATIVE FILE",
4 DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID) AS BLOCK
5 FROM summit.department;

ID ROWID                 OBJECT       RELATIVE FILE       BLOCK
-- ------------------    ---------    -------------       ---------
10 AAADHJAACAAAAORAAA    12745        2                   913
31 AAADHJAACAAAAORAAB    12745        2                   913
32 AAADHJAACAAAAORAAC    12745        2                   913
33 AAADHJAACAAAAORAAD    12745        2                   913
34 AAADHJAACAAAAORAAE    12745        2                   913
35 AAADHJAACAAAAORAAF    12745        2                   913
41 AAADHJAACAAAAORAAG    12745        2                   913
42 AAADHJAACAAAAORAAH    12745        2                   913
43 AAADHJAACAAAAORAAI    12745        2                   913
44 AAADHJAACAAAAORAAJ    12745        2                   913
45 AAADHJAACAAAAORAAK    12745        2                   913
50 AAADHJAACAAAAORAAL    12745        2                   913
12 rows selected.
```

Rechercher le numéro de fichier absolu

Lancez l'interrogation suivante pour obtenir les numéros de fichiers absolus des lignes dans SUMMIT.DEPARTMENT :

```
SQL> SELECT id, ROWID,
2 DBMS_ROWID.ROWID_TO_ABSOLUTE_FNO(ROWID,'SUMMIT','DEPARTMENT')
AS "FILE"
3 FROM summit.department;

ID         ROWID                FILE
---------  ------------------   ---------
10         AAADHJAACAAAAORAAA   2
31         AAADHJAACAAAAORAAB   2
32         AAADHJAACAAAAORAAC   2
33         AAADHJAACAAAAORAAD   2
34         AAADHJAACAAAAORAAE   2
35         AAADHJAACAAAAORAAF   2
41         AAADHJAACAAAAORAAG   2
42         AAADHJAACAAAAORAAH   2
43         AAADHJAACAAAAORAAI   2
44         AAADHJAACAAAAORAAJ   2
45         AAADHJAACAAAAORAAK   2
50         AAADHJAACAAAAORAAL   2
12 rows selected.
```


Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

2.Gestion d'index

2.1.Types d'index

2.1.1.Classification des index

Un index est une arborescence permettant d'accéder directement à une ligne spécifique dans une table. Les index peuvent être classifiés en fonction de leur conception ou de leur implémentation physique.

Avec une classification logique, les index sont regroupés du point de vue de l'application. Les différents types d'index logiques sont les suivants :

- index à une colonne,
- index concaténé,
- index unique,
- index non unique,
- index basé sur des fonctions,
- index de domaine.

Un index à une colonne comprend une seule colonne dans la clé d'index.
Par exemple, un index créé à partit de la colonne "empno" (numéro d'employé) de la table "emp" est un index à une colonne.

Un index concaténé, également appelé index composé, est créé à partit de plusieurs colonnes d'une table. Les colonnes d'in index concaténé ne doivent pas nécessairement être dans le même ordre que les colonnes de la table. En outre, ces colonnes ne sont pas forcément adjacentes.
Par exemple, un index créé à partir des colonnes "deptno" (numéro de département) et "job" (fonction) de la table "emp" est un index concaténé.
Un index concaténé peut contenir au maximum 32 colonnes. Toutefois, la taille cumulée de toutes les colonnes ne peut pas excéder un tiers de la taille du bloc de données.

Un index unique permet de s'assurer que deux lignes d'une table n'ont pas la même valeur dans la colonne qui définit l'index. Par conséquent, une clé d'index d'un index unique ne peut pointer que vers une seule ligne d'une table.

Dans un index non unique, une même clé peut avoir plusieurs lignes associées.

Un index basé sur des fonctions est créé lorsque est utilisé des fonctions ou des expressions qui appellent une ou plusieurs colonnes dans la table indexée. Ce type d'index pré calcule la valeur de la fonction ou de l'expression et le store dans l'index de la base de données. Les index basés sur des fonctions peuvent être crées comme B-tree ou index bitmap.

Les index de domaines sont des index propre à certaines application (Text, spatial…).

La classification physique des index dépend du format de stockage de ces index sur le disque. Les types d'index physiques sont les suivants :

- index partitionné,
- index B-tree,
- index bitmap.

Un index partitionné permet de stocker les entrées d'index correspondant à un index dans plusieurs segments. Ce type d'index est créé pour des tables volumineuses car cela permet de diviser un index volumineux en plusieurs unités facilement gérables. Un index partitionné permet de diminuer la

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

contention pour les recherches d'index. En effet, le partitionnement permet de répartir un index sur de nombreux tablespaces.
Les index partitionné sont souvent utilisés avec des tables partitionnées pour améliorer l'évolutivité et faciliter la gestion. Une partition d'index peut être créée pour chaque partition de table.

Un index B-tree est une arborescence ordonnée composée de nœuds d'index. Il contient une entrée par ligne.

Un index bitmap est un index physique contenant une entrée par groupe de lignes.

Le serveur Oracle gère tous les index lorsque des opérations DML sont exécutées sur une table. Les effets des différentes opérations DML sur les index B-tree sont les suivants :

Type d'opération DML	**Effet sur les index B-tree**
INSERT	Insertion de l'entrée d'index dans le bloc approprié
DELETE	Suppression logique de l'entrée d'index
UPDATE	Suppression logique de l'entrée d'index et insertion dans cet index

2.1.2.Index B-tree

Le processus d'E/S disque est le principal facteur déterminant la vitesse d'accès à une table. Pour réduire le temps d'E/S disque et améliorer les performances liées à l'accès aux tables, Oracle permet de créer des index de table. L'index B-tree est le type d'index le plus courant, et celui utilisé par défaut.
Un index B-tree stocke la valeur de clé et le ROWID de chaque ligne de la table. La structure d'un index B-tree comprend trois types de niveaux :

- la racine,
- les blocs de branche,
- les nœuds de feuille.

Au sommet d'un index B-tree se trouve la racine. Elle contient les entrées qui pointent vers le niveau suivant dans l'index.
Les blocs de branche constituent le niveau suivant d'un index B-tree. Ces blocs pointent vers les blocs du niveau suivant dans l'index.
Les nœuds de feuille constituent le niveau le plus bas d'un index B-tree. Ces nœuds contiennent les entrées d'index qui pointent vers les lignes d'une table. Les blocs feuille sont doublement liés pour que vous puissiez facilement parcourir l'index dans l'ordre croissant ou décroissant des valeurs de clé.

Une entrée d'index B-tree comporte les trois composants suivants :

- En-tête de l'entrée :

Stocke le nombre de colonnes et les informations de verrouillage.

- Couples longueur de colonne / valeur de clé :

Définit la taille d'une colonne dans une clé suivie de la valeur de la colonne.

- ROWID :

Indique le ROWID de la ligne contenant les valeurs de clé.

Un index B-tree a plusieurs caractéristiques :

- Il contient une entrée pour chaque ligne de la table, mais il ne contient pas d'entrée pour une ligne affichant une valeur NULL dans toutes les colonnes clé d'index.
- Si plusieurs lignes contiennent la même valeur de clé, ces valeurs de clé sont répétées.
- Dans un index non partitionné, toutes les lignes appartiennent au même segment. Par conséquent, un ROWID réduit est utilisé pour pointer vers les lignes de la table.

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

2.1.3.Index à clé inversée

Lors de l'insertion d'enregistrements par ordre croissant de valeur de clé, par exemple par numéro d'employé généré par le système, des goulets d'étranglement d'E/S peuvent se produire dans l'index. En effet, toutes les mises à jour d'index ont lieu au même endroit dans l'arborescence de l'index. Pour éviter ces goulets d'étranglement, Oracle propose l'index à clé inversée.

Un index à clé inversée a plusieurs fonctions. Tout d'abord, il inverse les octets de chaque valeur de la colonne indexée, à l'exception du ROWID. Par exemple, si le numéro d'employé 7698 est inséré dans la table "emp", la valeur de clé 8967 ests stockée dans l'index.
Une autre fonction de l'index à clé inversée : il répartit les mises à jour d'index sur des valeurs de clé contiguës dans l'arborescence de l'index.
Il existe une restriction relative à l'utilisation d'un index à clé inversé. On ne peut pas effectuer un balayage de l'intervalle de valeurs indexées sur un index à clé inversée.

2.1.4.Index bitmap

Un index bitmap est organisé comme un index B-tree. Toutefois, le nœud de feuille stocke une bitmap pour chaque valeur de clé. Il est préférable d'utiliser un index bitmap plutôt qu'un B-tree quand les requêtes utilisent une combinaison de clause WHERE avec l'opérateur OR. L'autre cas est si la clé des colonnes n'est pas souvent mise à jour ou en lecture seule.
Chaque bit de la bitmap correspond à un ROWID possible. En outre, si un bit est défini, cela signifie que la ligne comportant le ROWID correspondant contient également la valeur de clé.

Les composants d'un index bitmap sont :

- l'en-tête de l'entrée :
  Nombre de colonnes et informations de verrouillage
- la valeur de clé :
  Couples longueur / valeur pour chaque colonne de clé
- le ROWID de début :
  ROWID correspondante au premier bit de la bitmap
- le ROWID de fin :
  ROWID correspondante au dernier bit de la bitmap
- le segment bitmap :
  Chaîne de bits définis quand la ligne correspondante contient la valeur de clé.

Si des modifications sont apportées à la colonne clé d'une table, les bitmaps doivent être modifiées. Pour cela, les segments bitmap appropriés sont verrouillés. Etant donné que les verrous sont placés sur l'intégralité du segment bitmap, le bitmap ne peut être mis à jour par d'autres transactions tant que les verrous ne sont pas libérés.

Un index bitmap peut être utilisé dans les situations suivantes :

- Lorsqu'une table contient des millions de lignes et que la colonne clé a une cardinalité faible.
- Lorsque les requêtes utilisent souvent une combinaison de la condition WHERE multiple avec l'opérateur OR.
- Lorsque les colonnes clé sont en lecture seule ou ne sont pas fréquemment mises à jour.

2.1.5.Index bitmap ou index B-tree

Oracle offre la possibilité d'utiliser un index B-tree ou un index bitmap. Pour déterminer l'index approprié en fonction des cas, il est indispensable de connaître les différences entre ces deux types d'index.

B-tree	Bitmap
Convient pour les colonnes à cardinalité élevée	Convient pour les colonnes à faible cardinalité

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

Les mises à jour sur les colonnes clé sont relativement légères.	Les mises à jour sur les colonnes clé sont très lourdes.
Ne convient pas aux requêtes qui utilisent le prédicat OR.	Convient aux requêtes qui utilisent le prédicat OR.
Utile dans un environnement OLTP.	Utile dans un environnement DSS.

2.2.Création d'index

2.2.1.Contexte de création d'index

Avant de créer un index, il est nécessaire de connaître les points suivants :

- Les index augmentent les performances lors des requêtes utilisateurs mais ralentissent les ordres DML. Il convient toujours de minimiser les index sur les tables volatiles.
- Il ne faut jamais mettre les index dans les mêmes tablespaces que ceux contenant les rollback segments, segments temporaires, et les tables.
- Il peut être intéressant d'utiliser la clause NOLOGGING lors de la création de gros index pour gagner en performance.

2.2.2.Création d'un index B-tree

L'index B-tree est le type d'index le plus courant, et celui utilisé par défaut pour une table. Un index B-tree stocke des couples ROWID / valeur de clé. Ce type d'index peut être créé dans le compte de l'utilisateur auquel la table appartient ou dans un autre compte.
La syntaxe de la commande SQL permettant de créer un index B-tree est la suivante :

```
CREATE [UNIQUE] INDEX [schema.]index ON [schema.]table
(column [ASC|DESC][,column [ASC|DESC]]...)
[TABLESPACE tablespace_name]
[PCTFREE    integer]
[INITRANS   integer]
[MAXTRANS   interger]
[STORAGE    clause]
[LOGGING | NOLOGGING]
[NOSORT];
```

Exemple :

Créer un index B-tree normal EMP_IND1 appartenant au schéma SCOTT. Cet index doit être créé pour la table EMP dans le tablespace INDEX01 avec la valeur du paramètre PCTFREE à 25. Créer l'index à partir de la colonne EMPNO.

```
SQL> CREATE INDEX SCOTT.EMP_IND1 ON SCOTT.EMP(EMPNO)
  2  PCTFREE 25
  3  TABLESPACE INDEX01;
index created.
```

Pour créer un index de façon efficace, certaines règles doivent être prises en compte. Ces règles sont les suivantes :

- Trouver un compromis entre la requête et les besoins DML
- Placer l'index dans un tablespace distinct
- Utiliser des tailles d'extents uniformes
- Préférer l'option NOLOGGING pour les index volumineux
- Définir une valeur PCTFREE élevée si de nouvelles valeurs de clé doivent être ajoutées dans l'intervalle courant.

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

Un index peut être créé avec des propriétés semblables à celles d'un index existant avec l'outil Oracle Schema Manager.

2.2.3.Création d'un index à clé inversée

Pour éviter les goulets d'étranglement d'E/S, Oracle propose l'index à clé inversée. Un index à clé inversée inverse les octets de chaque colonne indexée.

```
CREATE [UNIQUE] INDEX [schema.]index ON [schema.]table
(column [ASC|DESC][,column [ASC|DESC]]...)
[TABLESPACE tablespace_name]
[PCTFREE    integer]
[INITRANS   integer]
[MAXTRANS   interger]
[STORAGE    clause]
[LOGGING | NOLOGGING]
REVERSE ;
```

Exemple :

Créer un index à clé inversée CUST_IND1 appartenant au schéma SCOTT et créé pour la table CUSTOMER. L'index est stocké dans le tablespace INDEX02 et est créé à partir de la colonne CUST_NO. La valeur du paramètre PCTFREE de l'index est de 30.

```
SQL> CREATE UNIQUE INDEX scott.cust_ind1
  2  ON scott.customer(cust_no) REVERSE
  3  PCTFREE 30
  4  TABLESPACE index02;
indexed created.
```

2.2.4.Création d'un index bitmap

Si les données impliquées ont une faible cardinalité, un index bitmap est plus approprié.

```
CREATE BITMAP INDEX [schema.]index ON [schema.]table
(column [ASC|DESC][,column [ASC|DESC]]...)
[TABLESPACE tablespace_name]
[PCTFREE    integer]
[INITRANS   integer]
[MAXTRANS   interger]
[STORAGE    clause]
[LOGGING | NOLOGGING]
[NOSORT];
```

Un index bitmap ne peut pas être unique.
Le paramètre d'initialisation CREATE_BITMAP_AREA_SIZE détermine la quantité d'espace utilisée pour stocker les segments bitmap dans la mémoire. La valeur par défaut de ce paramètre est 8 Mo. Une valeur supérieure à 8 Mo peut permettre d'accélérer la création de l'index.
Si la cardinalité est trop faible, l'administrateur peut affecter une valeur basse au paramètre CREATE_BITMAP_AREA_SIZE. Par exemple, si la cardinalité est seulement de deux, la valeur de ce paramètre peut être de l'ordre du kilo-octet plutôt que du mégaoctet. De manière générale, si la cardinalité est plus élevée, l'administrateur a besoin de davantage de mémoire pour obtenir des performances optimales.

Exemple :

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

Créer un index bitmap ORD_IND1 pour la table S_ORD sur la colonne ORDER_FILLED, stocké dans le tablespace INDEX01. La valeur du paramètre PCTFREE est de 25.

```
SQL> CREATE BITMAP INDEX SCOTT.ORD_IND1
  2  ON SCOTT.S_ORD (INV_NO)
  3  PCTFREE 25
  4  TABLESPACE INDEX01;
index created.
```

2.3.Gestion d'index

2.3.1.Modification des paramètres de stockage

Si un index vient à manquer d'espace de stockage, l'administrateur peut modifier les paramètres de stockage de cet index pour augmenter l'espace qui lui alloué.
Pour augmenter l'espace réservé à un index, l'administrateur peut augmenter le nombre maximal d'extents alloués à cet index.

Voici la syntaxe permettant de modifier les paramètres de stockage d'un index :

```
ALTER INDEX [schema.]index
[storage clause]
[INITRANS integer]
[MAXTRANS integer]
```

Exemple :

Pour augmenter l'espace disponible pour l'index EMP_IND1, il faut modifier ses paramètres de stockage en augmentant la valeur des paramètres MAXEXTENTS et MAXTRANS pour permettre à davantage d'extents et de transactions d'accéder à l'index simultanément.

```
SQL> ALTER INDEX SCOTT.EMP_IND1
  2  MAXTRANS 255
  3  STORAGE (MAXEXTENTS 200);
index altered.
```

2.3.2.Allocation et desallocation d'espace dans l'index

Un administrateur peut être amené à ajouter des extents à un index en prévision d'une forte activité d'insertion sur une table. L'objet d'extents évite l'extension dynamique des index et la dégradation des performances qui en résulte.

La syntaxe permettant d'allouer de l'espace à un index est la suivante :

```
ALTER INDEX [schema.]index
ALLOCATE EXTENT ( [SIZE integer [ K | M ] ] [DATAFILE
'filename'] )
```

Exemple :

Ajouter des extents à l'index EMP_IND1 du schéma SCOTT en allouant de l'espace.

```
SQL> ALTER INDEX scott.emp_ind1
  2  ALLOCATE EXTENT (SIZE 200K);
index altered.
```


Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

Cette commande affecte une taille de 200 K au nouvel extent de l'index EMP_IND1.

2.3.3.Libération dans l'index

De l'espace est libéré dans un index lorsque la table pour laquelle cet index a été créé est vidée. Pour limiter le gaspillage d'espace, il est possible de libérer manuellement l'espace inutilisé dans un index, au-dessus du high water mark.

La syntaxe permettant de libérer manuellement l'espace inutilisé dans un index au-dessus du high water mark est la suivante :

```
ALTER INDEX [schema.]index
[DATAFILE 'filename'] )
DEALLOCATE UNUSED [KEEP integer [ K | M ] ]
```

Exemple :

Libérer l'espace inutilisé au-dessus du high water mark dans l'index EMP_IND1 du schéma SCOTT.

```
SQL> ALTER INDEX scott.emp_ind1
  2  DEALLOCATE UNUSED;
index altered.
```

2.3.4.Situations de reconstruction d'un index

Une gestion efficace des index permet d'extraire plus rapidement les données souhaitées à partir d'un objet. La reconstruction des index est nécessaire à une gestion efficace des index.
Les index sont reconstruits dans différentes situations :

- Pour déplacer un index vers un autre tablespace

Cette opération peut être nécessaire si l'index se trouve dans le même tablespace que la table ou si des objets doivent être répartis sur plusieurs disques.

- Pour optimiser l'utilisation de l'espace en enlevant les entrées supprimées.

Ce problème se pose généralement pour les index en biais.
Exemple : un index a été créé à partir de la colonne ORDER_NO de la table ORDERS, dans laquelle les commandes exécutées sont supprimées et les nouvelles commandes sont ajoutées. Si quelques commandes anciennes sont encore en cours, la table peut contenir plusieurs blocs de feuille d'index comportant des entrées supprimées. La reconstruction de l'index permet alors d'améliorer l'utilisation de l'espace.

- Pour convertir un index à clé inversée existant en index B-tree normal ou inversement.

La commande permettant de reconstruire un index est la suivante :

```
ALTER INDEX [schema.]index REBUILD
[TABLESPACE tablespace]
[PCTFREE integer]
[INITRANS integer]
[MAXTRANS integer]
[storage clause]
[LOGGING | NOLOGGING]
[REVERSE | NOREVERSE];
```

Cette commande ne peut pas être utilisée pour convertir un index bitmap en index B-tree, ou vice-versa.

Exemple :

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

Reconstruire l'index EMP_IND1 appartenant au schéma SCOTT dans le tablespace INDEX01.

```
SQL> ALTER INDEX scott.emp_ind1 REBUILD
  2  TABLESPACE index01;
index altered.
```

2.3.5.Caractéristiques de reconstruction d'un index

Lorsque plusieurs modifications sont apportées aux données d'un index existant, cet index peut être soit supprimé puis recréé, soit reconstruit. La reconstruction d'un index prend moins de temps puisqu'une partie des données existantes est déjà triée. Le processus permettant de reconstruire un index est appelé reconstruction d'index.

Une reconstruction d'index permet de créer un nouvel index en utilisant un index existant comme source de données.
Une reconstruction d'index a certaines caractéristiques.

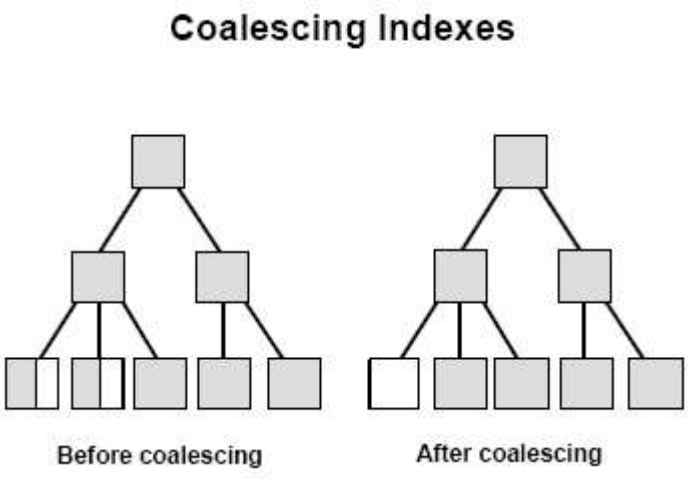
- Elle évite les opérations de tri car l'index est construit à partir d'un index existant. Cela permet d'obtenir de meilleures performances.
- L'ancien index est supprimé une fois la construction du nouvel index terminée. Lors d'une reconstruction d'index, il faut disposer de suffisamment d'espace pour stocker l'ancien et le nouvel index dans leurs tablespaces respectifs.
- Un index reconstruit ne contient pas d'éléments supprimés. Cela permet donc une utilisation plus efficace de l'espace.
- Lors d'une reconstruction d'index, les requêtes peuvent continuer à utiliser l'index existant pendant la construction du nouvel index.

2.3.6.Index coalescents

Lorsque l'on rencontre des index fragmenté, il est possible de les reconstruire, comme il a été vu, ou de les défragmenter. Avant de choisir l'un ou l'autre de ces options il est important de bien connaître les bénéfices de chacune de ces options et de sélectionner celle la plus adapter.
Il est possible de défragmenter un index grâce à la commande suivante :

```
SQL> ALTER INDEX hr.employye_idx COALESCE ;
   Index altered.
```

La figure ci-dessous montre les effets de la commande précédente sur l'index hr.employee_idx. Avant d'effectuer l'opération, les deux blocs de feuilles sont remplis à 50%. Après l'opération, le premier bloc est complètement rempli.



Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

2.3.7.Contrôle de la validité d'un index

Pour vérifier si un index est valide, tous ses blocs peuvent être analysés afin de s'assurer qu'ils ne sont pas corrompus. Cette opération met à jour la vue INDEX_STATS pour prendre en compte les informations relatives à l'index.
Ainsi, l'administrateur pourra réorganiser un index si il constate qu'il comporte beaucoup de lignes supprimées. Par exemple, un index peut être reconstruit si la proportion de colonnes DEL_LF_ROWS par rapport aux colonnes LF_ROWS excède 30 pour cent.

Avant d'interroger la vue INDEX_STATS, il faut la remplir.
Voici la commande SQL utilisée pour remplir cette vue :

```
ANALYSE INDEX [schéma.]index VALIDATE STRUCTURE;
```

Exemple :

Extraire les informations des colonnes NAME, BLOCKS, LF_ROWS et DEL_LF_ROWS concernant les index

```
SQL> SELECT name, blocks,lf_rows,del_lf_rows
  2  FROM INDEX_STATS;
NAME        BLOCKS LF_ROWS       DEL_LF_ROWS
------------------      ---------    ------------ ------------
EMP_IND1          5      14     0
```

2.3.8.Suppression d'un index

Les index qui ne sont utilisés que ponctuellement ne nécessitent pas d'être conservés. C'est le cas notamment d'un index créé pour un système OLTP dans lequel des requêtes ad hoc sont générées tous les ans ou tous les trimestres pour recueillir des informations en vue d'une réunion.

Lors de l'échec d'une instance Oracle, des index peuvent être marqués INVALID pour certains types d'opérations comme les chargements. Ces index peuvent être supprimées et recréés.

Un index peut être supprimé avant le chargement d'une grande quantité de données, puis recrée. Cela permet d'améliorer les performances du chargement et d'utiliser l'espace plus efficacement dans l'index.

La syntaxe de la commande SQL permettant de supprimer un index est la suivante :

```
DROP INDEX [schéma.]index;
```

2.3.9.Utilisation du monitoring pour identifier les index inutilisés

Avec la nouvelle version d'Oracle 9i, les statistiques sur l'usage des index peuvent être rassemblés et affichés par la vue dynamique V$OBJECT_USAGE. Par exemple, si les informations retournées montrent que l'index n'est jamais utilisé, celui-ci peut être supprimé, améliorant ainsi les performances lors d'ordres DML.
Chaque fois que la clause MONITORING USAGE est spécifiée, la vue dynamique précédente est réinitialisée pour l'index correspondant.

Exemple pour démarrer le monitoring sur un index :

```
SQL> ALTER INDEX summit.orders_id_ix
  2  MONITORING USAGE ;
     Index altered ;
```

Exemple pour arrêter le monitoring sur un index :

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

```
SQL> ALTER INDEX summit.orders_id_ix
  2  MONITORING USAGE ;
     Index altered ;
```

2.3.10.Informations sur les index

Les index de la base de données peuvent être surveillé en extrayant des informations générales sur ces index ou des informations relatives aux colonnes indexées à partir des vues DBA_INDEXES et DBA_IND_COLUMNS.

Une liste partielle des colonnes contenues dans la vue de dictionnaire de données DBA_INDEXES :

- OWNER
- INDEX_NAME
- INDEX_TYPE
- TABLE_OWNER
- TABLE_NAME
- UNIQUENESS
- TABLESPACE_NAME
- LOGGIG
- STATUS

Exemple :

Extraire des informations générales sur les index : interroger la vue DBA_INDEXES pour extraire les informations des colonnes INDEX_NAME, TABLESPACE_NAME et INDEX_TYPE pour tous les index appartenant au schéma SCOTT.

```
SQL> SELECT       index_name, tablespace_name, index_type
  2  FROM   dba_indexes
  3  WHERE        owner = 'SCOTT';

INDEX_NAME                     TABLESPACE_NAME           INDEX_TYPE
------------------------------ ------------------------- ----------
-
PK_DEPT                        USER_DATA                 NORMAL
PK_EMP                         USER_DATA                 NORMAL
S_CUSTOMER_ID_PK               USER_DATA                 NORMAL
S_DEPT_ID_PK                   USER_DATA                 NORMAL
S_DEPT_NAMPE_REGION_ID_UK      USER_DATA                 NORMAL
```

Une liste partielle des colonnes contenues dans la vue de dictionnaire de données DBA_IND_COLUMNS :

- INDEX_OWNER
- INDEX_NAME
- TABLE_OWNER
- TABLE_NAME
- COLUMN_NAME
- COLUMN_POSITION
- COLUMN_LENGTH

Exemple :

Extraire des informations sur les colonnes indexées : interroger la vue DBA_IND_COLUMNS pour extraire les colonnes INDEX_NAME,

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

TABLE_OWNER et TABLE_NAME. Rechercher les informations sur les index appartenant à SCOTT et les trier en fonction de la colonne INDEX_NAME.

```
SQL> SELECT       index_name, table_owner, table_name
  2  FROM   dba_ind_columns
  3  WHERE  index_owner = 'SCOTT'
  4  ORDER BY     index_name;
INDEX_NAME                     TABLE_OWNER                    INDEX_TYPE
------------------------------ ------------------------------ ----------
-
PK_DEPT                        SCOTT                          DEPT
PK_EMP                         SCOTT                          EMP
S_CUSTOMER_ID_PK               SCOTT                          S_CUSTOMER
S_DEPT_ID_PK                   SCOTT                          S_DEPT
S_DEPT_NAMPE_REGION_ID_UK      SCOTT                          S_DEPT
```

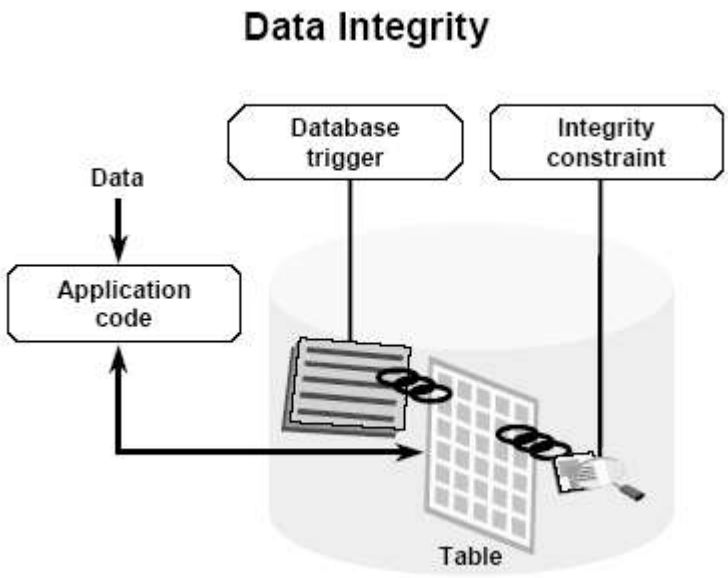

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

3.Gérer l'intégrité des données

Nous allons aborder dans ce chapitre les points suivants :

- Mise en œuvre des contraintes d'intégrité sur les données,
- Gestion des contraintes d'intégrité,
- Comment obtenir des informations sur les contraintes à partir du dictionnaire de données ?

3.1. Intégrité des données



3.1.1.Méthodes garantissant l'intégrité des données

L'intégrité des données garantit que les données d'une base de données respectent certaines règles. Il existe trois méthodes principales pour garantir l'intégrité des données :

- le code applicatif,
- les triggers de base de données,
- contraintes d'intégrité déclaratives.

Le choix de la méthode à utiliser pour appliquer ces règles relève de la conception de la base de données et incombe à son concepteur. L'administrateur de base de données est le premier concerné par la mise en œuvre de la méthode choisie par le concepteur et par l'équilibrage entre le besoin de performances et l'intégrité des données. Le code applicatif peut être mis en œuvre sous forme de procédures stockées dans la base de données ou sous forme d'applications exécutées sur le client.
Le présent chapitre porte sur l'utilisation des triggers de base de données et sur les contraintes d'intégrité.

Triggers de base de données

Les déclencheurs de base de données sont des programmes PL/SQL qui s'exécutent lorsqu'un événement donné, tel que l'insertion ou la mise à jour d'une colonne, se produit sur une table.
Vous pouvez activer ou désactiver les déclencheurs, c'est-à-dire que vous pouvez les définir pour qu'ils s'exécutent lorsque l'événement se produit, ou pour qu'ils ne s'exécutent pas, même s'ils sont définis. Les déclencheurs de base de données ne sont généralement créés que pour appliquer une règle complexe ne pouvant être définie sous la forme d'une contrainte d'intégrité.

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

Contraintes d'intégrité

Les contraintes d'intégrité constituent le mécanisme par excellence pour appliquer des règles, car :

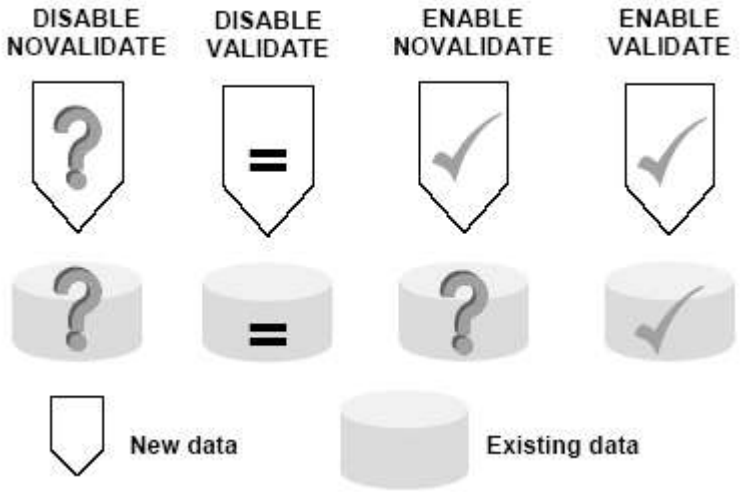
- elles améliorent les performances,
- elles sont simples à déclarer et à modifier, dans la mesure où elles nécessitent peu de code,
- elles centralisent les règles,
- elles sont souples (activées ou désactivées),
- elles sont entièrement documentées dans le dictionnaire des données.

Types de contrainte

Il existe cinq types de contrainte d'intégrité déclarative. Bien que les contraintes NOT NULL et CHECK ne nécessitent aucune attention particulière de la part de l'administrateur de base de données, les contraintes de clé primaire, de clé unique et de clé étrangère doivent être gérées pour garantir une disponibilité optimale et des performances acceptables.

Contrainte	Description
NOT NULL	Indique qu'une colonne ne peut contenir aucune valeur nulle
UNIQUE	Définit comme étant unique une colonne ou un groupe de colonnes.
PRIMARY KEY	Définit une colonne ou un groupe de colonnes comme clé primaire de la table.
FOREIGN KEY	Définit une colonne ou un groupe de colonne comme clé étrangère dans une contrainte d'intégrité référentielle.
CHECK	Définit une condition que chaque ligne de la table doit satisfaire.

Etats des contraintes



Une contrainte d'intégrité peut avoir l'un des états suivants :

- Disabled Novalidate,
- Disabled Validate,
- Enabled Novalidate ou Enforced,
- Enabled Validate.

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

Disabled Novalidate

Une contrainte ayant l'état Disabled Novalidate n'est pas vérifiée, bien que la définition de la contrainte existe toujours dans le dictionnaire de données. Les données de la table et les nouvelles données entrées ou mises à jour peuvent ne pas être conformes aux règles définies par la contrainte

Disabled Validate

Lorsqu'une contrainte a cet état, la modification des colonnes auxquelles s'applique la contrainte n'est pas autorisée. En outre, l'index de la contrainte est supprimé et la contrainte est désactivée.
Pour une contrainte unique, cet état permet de charger efficacement les données d'une table non partitionnée dans une table partitionnée en utilisant l'option EXCHANGE PARTITION de la commande ALTER TABLE.

Enabled Novalidate (Enforced)

Cet état de contrainte interdit l'entrée des données qui violent les règles de la contrainte. Toutefois, la table peut contenir des données non valides, c'est-à-dire qui violent la contrainte. En général, il s'agit d'un état intermédiaire qui garantit que toutes les nouvelles données ont été vérifiées avant d'être acceptées dans la table.

Enabled Validate

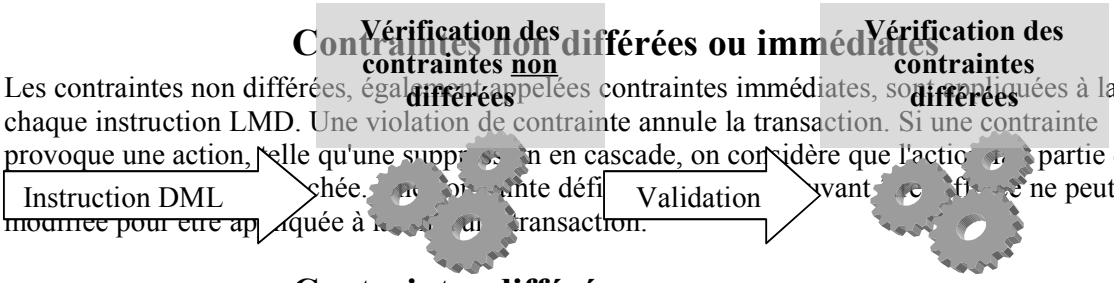
Lorsqu'une contrainte a cet état, cela implique que toutes les données de la table sont conformes à la contrainte. En outre, cet état empêche d'entrer des données non valides dans la table. Il s'agit de l'état normal d'une contrainte de traitement de transaction "online".Lorsqu'une contrainte passe de l'état Disabled à l'état Enabled Validate, la table est verrouillée et la conformité de toutes les données de la table est vérifiée. Cette situation peut provoquer le report de l'exécution d'opérations LMD, telles que le chargement de données. Il est donc **conseillé** de faire d'abord **passer la contrainte de l'état Disabled à l'état Enable Novalidate, puis à l'état Enable Validate**.

Contraintes différées

Vous pouvez définir le point d'une transaction à partir duquel une contrainte sera vérifiée en définissant la contrainte de manière appropriée.

Contraintes non différées ou immédiates

Les contraintes non différées, également appelées contraintes immédiates, sont appliquées à la fin de chaque instruction LMD. Une violation de contrainte annule la transaction. Si une contrainte provoque une action, telle qu'une suppression en cascade, on considère que l'action fait partie de l'instruction effectuée. Une contrainte définie comme ne pouvant pas être différée ne peut pas être modifiée pour être appliquée à la fin d'une transaction.



Contraintes différées

Les contraintes différées ne sont vérifiées que lorsqu'une transaction est validée. Si une violation de contrainte est détectée lors de la validation, l'ensemble de la transaction est annulé. Ces contraintes s'avèrent très utiles lorsque les lignes parent et les lignes enfant d'une relation de clé étrangère sont entrées simultanément, comme dans le cas d'un système de traitement des commandes où les articles et la commande sont entrés simultanément.
Pour qu'une contrainte puisse être différée, elle doit être définie lors de sa création comme pouvant être différée. Une contrainte définie comme pouvant être différée peut être indiquée comme suit :

- *Initially immediate* : indique qu'il s'agit par défaut d'une contrainte immédiate, sauf indication contraire explicite.

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

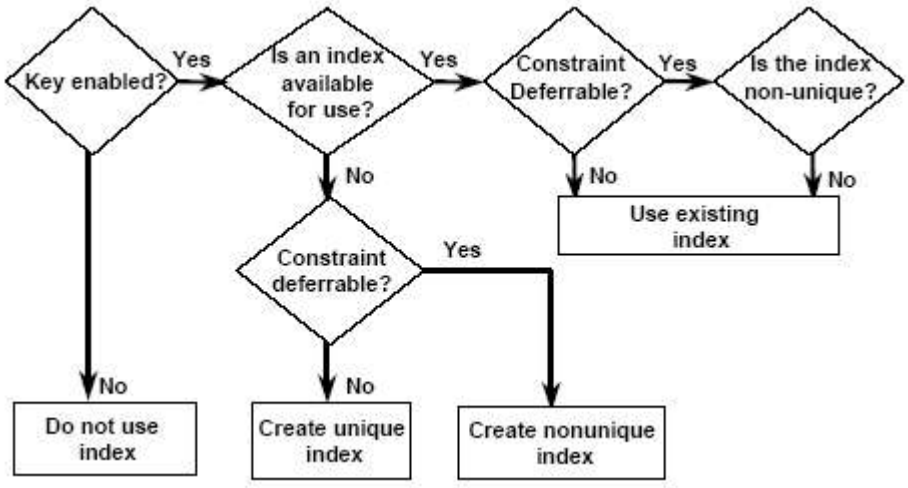
- *Initially deferred* : indique que, par défaut, la contrainte ne doit être appliquée qu'à la fin de la transaction.

Modifier l'application des contraintes

Bien que le mode d'application par défaut d'une contrainte pouvant être différée soit défini et stocké dans le dictionnaire de données, les applications peuvent modifier la contrainte pour qu'elle fonctionne en différé ou en immédiat. Pour cela, il faut utiliser la commande ALTER SESSION ou SET CONSTRAINT :

```
ALTER SESSION
SET CONSTRAINT[S] =
{IMMEDIATE|DEFERRED|DEFAULT}
SET CONSTRAINT[S]
{constraint [, constraint ]...
|ALL }
{IMMEDIATE|DEFERRED}
```

Appliquer des contraintes de clé primaire et de clé unique



Les contraintes de clé primaire et de clé unique sont appliquées en utilisant des index. Vous pouvez contrôler l'emplacement et le type de l'index utilisé pour appliquer ces contraintes.
Le serveur Oracle utilise la procédure suivante pour mettre en œuvre les contraintes de clé primaire et de clé unique :

- Si la contrainte est désactivée, aucun index n'est nécessaire.
- Si la contrainte est activée et que les colonnes de la contrainte forment la première partie d'un index, l'index est utilisé pour appliquer la contrainte.
- Si la contrainte est activée et que les colonnes de la contrainte ne correspondent pas au début d'un index, un index portant le nom de la contrainte est créé selon les règles suivantes :
- S'il s'agit d'une clé pouvant être différée, un index non unique est créé dans la colonne de la clé.
- S'il s'agit d'une clé ne pouvant pas être différée, un index unique est créé.
- Si un index est utilisable et que la contrainte ne peut pas être différée, il faut utiliser l'index existant. Si la contrainte peut être différé et que l'index n'est pas unique, il faut aussi utiliser l'index existant.

Considération sur les clés étrangères

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

Action souhaitée	Solution
Supprimer la table mère	Contraintes en cascade
Tronquer la table mère	Désactiver ou supprimer la clef étrangère
Supprimer un tablespace contenant la table mère	Utiliser la clause CASCADE CONSTRAINTS
Eviter de verrouiller la table enfant lors d'opérations DML dans la table mère	Créer un index sur la clef étrangère
Exécuter des opérations DML dans la table enfant	Mettre en ligne le tablespace contenant l'index de la clef parent

Remarques sur l'utilisation des contraintes de clés étrangères

Vous devez tenir compte de plusieurs éléments pour gérer des tables dans une relation de clé étrangère.

DDL impliquant la table mère

La clé étrangère doit être supprimée avant la table mère. Utilisez la commande suivante pour exécuter les deux actions à l'aide d'une seule instruction :

```
DROP TABLE table CASCADE CONSTRAINTS
```

Vous ne pouvez pas tronquer la table mère sans supprimer ou désactiver la clé étrangère.

Vous devez supprimer la clé étrangère pour pouvoir supprimer le tablespace contenant la table mère. Pour ce faire, utilisez la commande suivante :

```
DROP TABLESPACE tablespace INCLUDING CONTENTS
CASCADE CONSTRAINTS
```

Opérations DML sur les tables dans une relation de clé étrangère

Si l'option DELETE CASCADE n'est pas utilisée lors de la suppression des lignes de la table mère, le serveur Oracle doit vérifier dans la table enfant qu'il n'y a pas de ligne contenant la clé étrangère correspondante.
De même, la clé parent ne peut être mise à jour que si l'ancienne clé ne figure dans aucune ligne enfant**. Si la clé étrangère de la table enfant ne contient pas d'index, le serveur Oracle verrouille la table enfant** et interdit les modifications pour garantir l'intégrité référentielle. Si la table contient un index, l'intégrité référentielle est conservée en verrouillant les entrées d'index et en évitant d'utiliser des verrous plus restrictifs sur la table enfant. Si les deux tables doivent être mises à jour simultanément par des transactions différentes, créez un index dans les colonnes de la clé étrangère. Lorsque les données sont insérées ou que la colonne de la clé étrangère de la table enfant est mise à jour, le serveur Oracle vérifie l'index de la table mère utilisée pour appliquer la clé référencée. Par conséquent, l'opération n'aboutit que si le tablespace contenant l'index est "online". Notez que le tablespace contenant la table mère ne doit pas nécessairement être "online" pour exécuter des opérations DML sur la table enfant.
Remarque : il est recommandé de créer des index dans les colonnes de la clé étrangère, car cela apporte d'autres avantages en matière de performances.

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

3.2. Mettre en œuvre des contraintes

3.2.1. Définir les contraintes lors de la création d'un table

Vous pouvez définir une contrainte lorsque vous créez une table ou lorsque vous la modifiez. Il suffit d'utiliser la clause CONSTRAINT pour définir une contrainte. Pour créer une contraintes d'intégrité référentiel (Primary Key, Foreign Key), la table parent doit appartenir à votre schéma, ou vous devez posséder les privilèges REFERENCES sur les colonnes de la table parent où la clé est référencée.

```
CREATE TABLE summit.employee(
id NUMBER(7)
     CONSTRAINT employee_id_pk PRIMARY KEY
     DEFERRABLE
     USING INDEX
            STORAGE(INITIAL 100K NEXT 100K)
            TABLESPACE indx,
last_name VARCHAR2(25)
     CONSTRAINT employee_last_name_nn NOT NULL,
dept_id NUMBER(7))
TABLESPACE data;
```

3.2.2. Syntaxe pour les contraintes de colonne ou de table

Lors de la création de la table, vous pouvez créer la contrainte en utilisant la syntaxe suivante pour définir la colonne :

```
column datatype [CONSTRAINT constraint_name]
in_line_constraint
[defer_spec]

in_line_constraint :==
{[NOT] NULL
|PRIMARY KEY [USING INDEX index_clause]
|UNIQUE [USING INDEX index_clause]
|REFERENCES [schema.]table [(column)]
[ON DELETE CASCADE]
|CHECK (condition)
}

defer_spec :==
[NOT DEFERRABLE|DEFERRABLE [INITIALLY {IMMEDIATE|DEFERRED}]
]
[DISABLE|ENABLE [VALIDATE|NOVALIDATE]]
```

où	CONSTRAINT_NAME	identifie la contrainte d'intégrité par le nom *constraint* stocké dans le dictionnaire de données,
	USING INDEX	indique que les paramètres définis dans *index-clause* doivent être utilisés pour l'index auquel fait appel le serveur Oracle afin d'appliquer une contrainte de clé unique ou primaire (l'index porte le même nom que la contrainte),
	DEFERRABLE	indique que la vérification de la contrainte peut être reportée à la fin de la transaction à l'aide de la commande SET CONSTRAINT(S),
	NOT DEFERRABLE	indique que la contrainte est vérifiée à la fin de chaque instruction LMD (une contrainte NOT DEFERRABLE ne peut pas être différée par les sessions ou par les transactions. NOT DEFERRABLE est utilisé par défaut).
	INITIALLY IMMEDIATE	indique qu'au début de chaque transaction, la contrainte doit, par défaut, être vérifiée à la fin de chaque instruction LMD (si aucune clause INITIALLY n'est définie, INITIALLY IMMEDIATE est utilisé par défaut),

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

	INITIALLY DEFERRED	indique qu'il s'agit d'une contrainte DEFERRABLE et que, par défaut, elle n'est vérifiée qu'à la fin de chaque transaction,
	DISABLE	désactive la contrainte d'intégrité (lorsqu'une contrainte d'intégrité est désactivée, le serveur Oracle ne l'applique pas).

3.2.3.Syntaxe : contrainte de table

Vous pouvez également créer la contrainte de table à l'aide de la syntaxe suivante :

```
[CONSTRAINT constraint]
out_of_line_constraint

out_of_line_constraint :==
{PRIMARY KEY (column [, column ]... )
[USING INDEX index_clause]
|UNIQUE (column [, column ]... )
[USING INDEX index_clause]
|FOREIGN KEY (column [, column ]... )
REFERENCES [schema.]table [(column [, column ]... )]
[ON DELETE CASCADE]
|CHECK (condition)
}
[deferrable_specification]
```

Remarque :
Il est recommandé d'adopter une convention d'appellation standard pour les contraintes, notamment avec les contraintes CHECK, puisque vous pouvez créer plusieurs fois la même avec des noms différents.
Dans les cas suivants, vous devez utiliser des contraintes de table :

- lorsqu'une contrainte s'applique à plusieurs colonnes,
- lorsqu'une table est modifiée pour ajouter des contraintes autres que NOT NULL.

3.2.4.Définir des contraintes après la création d'une table :

Exemple :

```
ALTER TABLE summit.employee
ADD(CONSTRAINT employee_dept_id_fk FOREIGN KEY(dept_id)
REFERENCES summit.department(id)
DEFERRABLE INITIALLY DEFERRED);
```

Remarque : la clause EXCEPTIONS, décrite dans la section "Activer les contraintes" de ce chapitre, peut être utilisée pour identifier les lignes qui violent une contrainte ajoutée à l'aide de la commande ALTER TABLE.

3.2.5.Instructions de définition des contraintes

Tenez compte des points suivants pour définir des contraintes :

- Placez les index utilisés pour appliquer les contraintes de clé primaire et de clé unique dans un tablespace différent de celui de la table. Pour ce faire, vous pouvez soit définir la clause USING INDEX, soit créer la table et l'index, puis modifier la table pour ajouter ou activer la contrainte.
- Si les données sont souvent chargées en masse dans une table, il est préférable de désactiver les contraintes, de charger les données, puis de réactiver les contraintes. Si vous utilisez un index unique pour appliquer une contrainte de clé primaire ou de clé unique, cet index doit être supprimé lorsque vous désactivez la contrainte. Dans une telle situation, vous pouvez améliorer les performances en utilisant un index non unique pour appliquer des contraintes de

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

clé primaire ou de clé unique. Pour ce faire, créez une clé pouvant être différée, ou créez l'index avant de définir ou d'activer la clé.

- Si une table contient une clé étrangère d'auto référencement, utilisez l'une des méthodes suivantes pour charger les données :
  - Définissez ou activez la clé étrangère après le premier chargement de données.
  - Définissez la contrainte comme pouvant être différée. La seconde méthode s'avère très utile lorsque des données sont chargées fréquemment.

3.3. Gérer les contraintes

3.3.1.Activer les Contraintes

Enable NOVALIDATE

Enable NOVALIDATE	• Ne verrouille pas la table • Les clés primaires doivent utiliser des index non uniques

```
ALTER TABLE summit.department
ENABLE NOVALIDATE CONSTRAINT dept_pk;
```

Une contrainte désactivée peut être activée de deux manières : à l'aide de l'option **enable NOVALIDATE** ou **enable VALIDATE**.

L'activation d'une contrainte de non-validation est beaucoup plus rapide que l'activation d'une contrainte de validation, car si la contrainte peut être différée, il n'y a pas de détection de violation de contrainte sur les données existantes. Si vous utilisez cette option pour activer une contrainte, il n'est pas nécessaire de verrouiller la table. Utilisez cette méthode lorsque de nombreuses opérations LMD sont exécutées sur la table, comme c'est le cas dans un environnement OLTP.

Syntaxe

Utilisez la commande suivante pour activer une contrainte de non-validation :

```
ALTER TABLE [ schema. ] table
ENABLE NOVALIDATE {CONSTRAINT constraint
| PRIMARY KEY
| UNIQUE ( column [, column ] ... ) }
[ USING INDEX index_clause ]
```

Restrictions

La clause USING INDEX ne s'applique qu'aux contraintes de clé primaire ou de clé unique créées comme contraintes pouvant être différées, et seulement si l'une des conditions suivantes est vraie :

- les contraintes n'ont pas été activées lors de leur création,
- les contraintes ont été désactivées et l'index a été supprimé.

Toutefois, si vous devez créer l'index, cette méthode d'activation de contrainte ne procure aucun avantage particulier par rapport à la méthode d'activation de la validation, car le serveur Oracle verrouille la table pour créer l'index.
Remarque : la désactivation des contraintes est décrite dans le cours *Introduction to SQL and PL/SQL.*

Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

Enable VALIDATE

Enable VALIDATE	• Ne Verrouille pas la table • Peut utiliser des index uniques ou non uniques • Nécessite des données de table valides

```
ALTER TABLE summit.employee
ENABLE VALIDATE CONSTRAINT emp_dept_fk;
```

L'activation d'une contrainte de validation déclenche une détection de violation de contrainte sur les données existantes. Cette action est effectuée par défaut lorsqu'une contrainte est activée. Si la détection est lancée alors que la contrainte est désactivée, les effets sont les suivants :

- La table est verrouillée et aucune modification n'est appliquée tant que la validation des données existantes n'est pas terminée.
- Le serveur Oracle crée un index s'il n'en existe pas dans les colonnes d'index. Il crée un index unique lors de l'activation d'une contrainte de clé primaire ou de clé unique ne pouvant être différée. Il crée un index non unique pour une contrainte de clé primaire ou de clé unique pouvant être différée.

Si cette commande est exécutée lorsqu'une contrainte est en vigueur, la table n'a pas besoin d'être verrouillée durant la validation. La contrainte appliquée vérifie qu'aucune donnée insérée lors de la validation ne viole ces règles. Les avantages sont les suivants :

- Toutes les contraintes sont activées simultanément.
- Chaque contrainte est exécutée en interne et en parallèle.
- Les opérations simultanées sur la table sont rendues possibles.

Syntaxe

Utilisez la commande suivante pour activer une contrainte de validation :

```
ALTER TABLE [ schema. ] table
ENABLE [ VALIDATE ]{CONSTRAINT constraint
| PRIMARY KEY
| UNIQUE ( column [, column ] ... ) }
[ USING INDEX index_clause ]
[ EXCEPTIONS INTO [ schema. ] table ]
```

Remarque :
• VALIDATE est l'option par défaut. Il est inutile de la définir pour activer une contrainte désactivée.
• Si des données de la table violent la contrainte, l'instruction est annulée et la contrainte reste désactivée.

3.3.2.Utiliser la table EXCEPTIONS

1. Créez la table EXCEPTIONS (utlexcpt.sql).
2. Exécutez la commande ALTER TABLE avec la clause EXCEPTIONS.
3. Utilisez l'interrogation imbriquée sur la table EXCEPTIONS pour rechercher les lignes contenant des valeurs non valides.
4. Rectifiez les erreurs.
5. Réexécutez la commande ALTER TABLE pour activer la contrainte.

Identifier une violation de contrainte due aux lignes

La clause EXCEPTIONS permet d'identifier les lignes qui violent une contrainte activée. Pour identifier les violations de contrainte, les rectifier et réactiver une contrainte, procédez comme suit :

1. Si la table d'exceptions n'a pas encore été créée, exécutez le script utlexcpt.sql du répertoire ADMIN :

```
SQL> @?/rdbms/admin/utlexcpt
```


Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

```
Statement processed.
SQL> DESCRIBE exceptions
Name                          Null?    Type
----------------------------- -------  ----------------
ROW_ID                                 UNDEFINED
OWNER                                  VARCHAR2(30)
TABLE_NAME                                    VARCHAR2(30)
CONSTRAINT                                    VARCHAR2(30)
```

Sous Windows NT, ce script se trouve dans le répertoire `%ORACLE_HOME%\RDBMS\ADMIN.`

2. Exécutez la commande ALTER TABLE en utilisant la clause EXCEPTIONS :

```
SQL> ALTER TABLE summit.employee
2 ENABLE VALIDATE CONSTRAINT employee_dept_id_fk
3 EXCEPTIONS INTO system.exceptions;
ALTER TABLE summit.employee
*
ORA-02298: cannot enable (summit.EMP_DEPT_FK) - parent keys not
found
```

Si la table d'exceptions n'est pas qualifiée avec le nom du propriétaire, elle doit appartenir au propriétaire de la table en cours de modification. Des lignes sont insérées dans la table d'exceptions. Si vous réexécutez la commande, tronquez la table d'exceptions pour supprimer toutes les lignes existantes.

3. Identifiez les données non valides en lançant une sous-interrogation sur la table EXCEPTIONS :

```
SQL> SELECT rowid, id, last_name, dept_id
2 FROM summit.employee
3 WHERE ROWID in (SELECT row_id
4 FROM exceptions)
5 FOR UPDATE;
ROWID              ID    LAST_NAME       DEPT_ID
------------------ ----- --------------- --------
AAAAeyAADAAAAA1AAA  1003  Pirie            50
1 row selected.
```

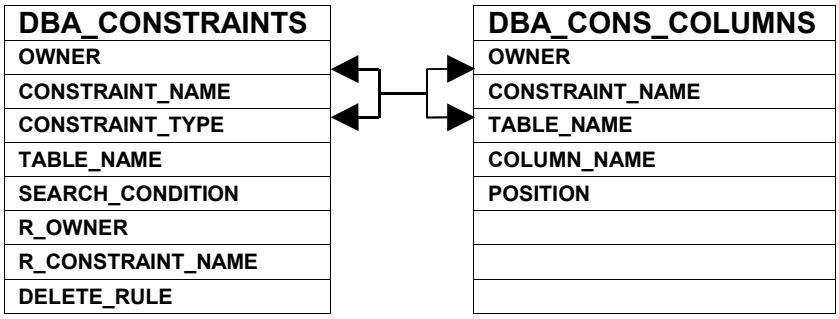
4. Corrigez les erreurs de données :

```
SQL> UPDATE summit.employee
2 SET id=10
3 WHERE rowid='AAAAeyAADAAAAA1AAA';
1 row processed.
SQL> COMMIT;
Statement processed.
```

5. Tronquez la table exceptions et réactivez la contrainte :

```
SQL> TRUNCATE TABLE exceptions;
Statement processed.
SQL> ALTER TABLE summit.employee
2 ENABLE VALIDATE CONSTRAINT employee_dept_id_fk
3 EXCEPTIONS INTO system.exceptions;
Statement processed.
```

Obtenir des informations sur les contraintes



Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

STATUS	
DEFERRABLE	
DEFERRED	
VALIDATED	
GENERATED	
BAD	
RELY	
LAST_CHANGE	

Contraintes et statut

En lançant l'interrogation suivante pour obtenir le nom, le type et le statut de toutes les contraintes de la table EMPLOYEE de l'utilisateur SUMMIT :

```
SQL> SELECT constraint_name, constraint_type, deferrable,
     2 deferred, validated
     3 FROM dba_constraints
     4 WHERE owner='SUMMIT'
     5 AND table_name='EMPLOYEE';

CONSTRAINT_NAME   C DEFERRABLE             DEFERRED    VALIDATED
----------------  - --------------- ----------- ----------
EMPLOYEE_DEPT..   R DEFERRABLE             DEFERRED    VALIDATED
EMPLOYEE_ID_PK    P DEFERRABLE             IMMEDIATE   VALIDATED
SYS_C00565              C NOT DEFERRABLE   IMMEDIATE   VALIDATED

3 rows selected.
```

Le tableau suivant montre les colonnes non explicites de la vue DBA_CONSTRAINTS.

Nom	Description
CONSTRAINT_TYPE	Le type de la contrainte est P, U, R ou C selon qu'il s'agit respectivement d'une contrainte de clé primaire, de clé unique, de clé étrangère ou d'une contrainte CHECK. Les contraintes NOT NULL sont stockées sous forme de contraintes CHECK.
SEARCH_CONDITION	Indique la condition définie pour une contrainte CHECK.
R_OWNER R_CONSTRAINT_NAME	Indique le propriétaire et le nom de la contrainte référencée pour les clés étrangères.
GENERATED	Indique si le nom de contrainte a été généré par le système (les valeurs acceptées sont "USER NAME" et "GENERATED NAME").
BAD	Indique que la contrainte doit être réécrite pour éviter certaines situations, telles que les problèmes de compatibilité avec l'an 2000 (ceci peut arriver, car les versions antérieures d'Oracle autorisaient les années à 2 chiffres dans les contraintes CHECK).
RELY	Si cet indicateur est défini, il est utilisé dans l'optimiseur.
LAST_CHANGE	Date de la dernière activation ou désactivation de la contrainte.

Colonnes des contraintes

Pour obtenir les colonnes des contraintes de la table EMPLOYEE de l'utilisateur SUMMIT, lancez l'interrogation suivante :

```
SQL> SELECT c.constraint_name, c.constraint_type,
     2 cc.column_name
     3 FROM dba_constraints c, dba_cons_columns cc
     4 WHERE c.owner='SUMMIT'
     5 AND c.table_name='EMPLOYEE'
     6 AND c.owner = cc.owner
```


Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs

```
     7 AND c.constraint_name = cc.constraint_name
     8 ORDER BY cc.position;

CONSTRAINT_NAME     C COLUMN_NAME
------------------- - ---------------
EMPLOYEE_DEPT...    R DEPT_ID
EMPLOYEE_ID_PK      P ID
SYS_C00565                C LAST_NAME

3 rows selected.
```

Rechercher les relations entre les clés primaires et les clés étrangères

Pour obtenir les clés étrangères de la table EMPLOYEE de l'utilisateur SUMMIT et les contraintes parent, lancez l'interrogation suivante :

```
SQL> SELECT c.constraint_name AS "Foreign Key",
     2 p.constraint_name AS "Referenced Key",
     3 p.constraint_type,
     4 p.owner,
     5 p.table_name
     6 FROM dba_constraints c, dba_constraints p
     7 WHERE c.owner='SUMMIT'
     8 AND c.table_name='EMPLOYEE'
     9 AND c.constraint_type='R'
     10 AND c.r_owner=p.owner
     11 AND c.r_constraint_name = p.constraint_name;

Foreign Key        Referenced  Key    C  OWNER       TABLE_NAME
-----------        ---------------    -  -----       ---------
EMPLOYEE_DEPT..     DEPT_PK           P  HR          DEPARTMENT


1 row selected.
```


Ce document est la propriété de Supinfo et est soumis aux règles de droits d'auteurs