Qu'est Qu'un Tablespace ?

Un tablespace est un espace logique qui contient les objets stockés dans la base de données comme les tables ou les indexes.
Un tablespace est composé d'au moins un datafile, c'est à dire un fichier de données qui est physiquement présent sur le serveur à l'endroit stipulé lors de sa création.
Chaque datafile est constitué de segments d'au moins un extent (ou page) lui-même constitué d'au moins 3 blocs : l'élément le plus petit d'une base de données.
L'extent n'a aucune signification particulière, c'est juste un groupe de blocs contigus pouvant accueillir des données, nous verrons néanmoins que cette notion d'extent peut poser des problèmes de gestion d'espace disque.

1.1. Les différents types de tablespaces

a. Le tablespace temporaire

Un tablespace temporaire est un tablespace spécifique aux opérations de tri pour lesquelles la SORT_AREA_SIZE ne serait pas suffisamment grande.
Ce tablespace n'est pas destiné à accueillir des objets de la base de données et son usage est réservé au système.
Depuis la version 9i, Oracle permet de définir un tablespace par défaut au niveau base de données (à la création de la base) ou utilisateur, chaque utilisateur pouvant avoir son propre tablespace temporaire ce qui est particulièrement pratique s'il existe un utilisateur spécifique pour les gros batchs par exemple.

Paramètrage du tablespace temporaire de la base

```
CREATE DATABASE <SID>...
   DEFAULT TEMPORARY TABLESPACE temp;
```

Modification du tablespace temporaire de la base

```
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE temp2;
```

Création d'un tablespace temporaire

```
CREATE TEMPORARY TABLESPACE temp
   TEMPFILE 'g:\oracle\oradata\orafrance\temp01.dbf'
   SIZE 20M EXTENT MANAGEMENT LOCAL UNIFORM SIZE 10M;
```

Assignation d'un tablespace temporaire à un utilisateur

```
ALTER USER orafrance
   TEMPORARY TABLESPACE temp;
```

b. Le tablespace UNDO

Le tablespace UNDO, comme son nom l'indique, est réservé exclusivement à l'annulation des

commandes DML (UPDATE, INSERT, etc...).
Lorsqu'on exécute l'ordre DELETE par exemple, Oracle commence par copier les lignes à supprimer dans le tablespace UNDO et ensuite indique que les blocs contenant les données dans le tablespace d'origine sont libres.
Un ROLLBACK permettre de revenir en arrière alors que le COMMIT supprimera les lignes du tablespace UNDO (on comprend mieux ici pourquoi un DELETE est si long : 2 écritures pour une suppression :-/).
Le tablespace UNDO est unique à une base de données et est guidé par les paramètres suivants du fichier d'initialisation de la base :

UNDO MANAGEMENT	AUTO ou MANUAL	Laisse le système gérer automatiquement le segment d'annulation ou permet d'utiliser le UNDO comme les ROLLBACK SEGMENT des versions antérieures.
UNDO TABLESPACE	nom du tablespace	Indique le nom du tablespace.
UNDO_SUPPRESS_ERRORS	TRUE ou FALSE	Permet d'ignorer ou non les erreurs liées à une gestion inappropriée du tablespace UNDO (i.e. commande prévue pour les rollback segments sur le UNDO : ALTER ROLLBACK...SET TRANSACTION USE ROLLBACK...).
UNDO_RETENTION	nombre de secondes	Durée de rétention des données dans le tablespace UNDO

Paramètrage du tablespace UNDO de la base

```
CREATE DATABASE <SID>...
   UNDO TABLESPACE undotbs
   DATAFILE 'g:\oracle\oradata\orafrance\undotbs.dbf' size 100M;
```

Création du tablespace UNDO

```
CREATE UNDO TABLESPACE undotbs
   DATAFILE 'g:\oracle\oradata\orafrance\undotbs.dbf' size 100M;
```

Modification du tablespace UNDO utilisé

```
ALTER SYSTEM
   SET UNDO_TABLESPACE=undotbs2;
```

c. Le tablespace transportable

Le tablespace transportable, introduit dans la version 8i, sert à copier les données entre deux bases de données.
Depuis la 9i, la taille des blocs de la base ne doit plus être nécessairement identique.
Dans l'exemple suivant nous allons copier les tablespaces **DATA_TBS** et **INDEX_TBS** de la base

dvp1 vers **dvp2**.
Pour qu'un tablespace puisse être transporté, il doit contenir tous les objets interdépendants. On ne pourra par exemple pas transporter un tablespace qui contient une table dont les indexes seraient créés dans un autre tablespace.

ATTENTION : les tablespaces ne sont pas transportables dans n'importe quelle condition :

- Les bases de données source et cible doivent être sur des plateformes identiques, impossible donc de transporter un tablespace d'un serveur Sun Solaris à un serveur Windows 2000.
- La source et la cible doivent utiliser le même jeu de caractère (character set et national character set).
- Il est évidemment impossible de transporter un tablespace dans une base qui contiendrait un tablespace de même nom.
- Enfin, les tablespaces transportables ne supportent pas : les vues matérialisés et les indexes de function

Création des tablespaces

```
SQL> create tablespace DATA_TBS
   2 datafile 'd:\dvp1\oradata\data_tbs.ora' size 1M reuse
   3 AUTOEXTEND OFF
   4 ONLINE
   5 default storage
   6 (initial 32 k next 32 k
   7 minextents 2 maxextents unlimited
   8 pctincrease 1);

Tablespace created.

SQL> create tablespace INDEX_TBS
   2 datafile 'd:\dvp1\oradata\index_tbs.ora' size 1M reuse
   3 AUTOEXTEND OFF
   4 ONLINE
   5 default storage
   6 (initial 32 k next 32 k
   7 minextents 2 maxextents unlimited
   8 pctincrease 1) ;

Tablespace created.
```

Création de la table et sa Primary Key

```
SQL> create table orafrance.matable (col1 NUMBER, col2 NUMBER )
   2 tablespace DATA_TBS;

Table created.

SQL> alter table orafrance.matable add
  2 (constraint matable_pk primary key (col1)
  3 USING INDEX TABLESPACE INDEX_TBS
  4 );

Table altered.
```

Ici, la clé primaire est créée dans le tablespace INDX_TBS alors que la table est dans DATA_TBS.

Vérification de la transportabilité des tablespaces

```
SQL> execute DBMS_TTS.TRANSPORT_SET_CHECK ('DATA_TBS',TRUE);

PL/SQL procedure successfully completed.

SQL> SELECT * FROM TRANSPORT_SET_VIOLATIONS;

no rows selected

SQL> execute DBMS_TTS.TRANSPORT_SET_CHECK ('INDEX_TBS',TRUE);

PL/SQL procedure successfully completed.

SQL> SELECT * FROM TRANSPORT_SET_VIOLATIONS;

VIOLATIONS
--------------------------------------------------------------------------------
Index ORAFRANCE.MATABLE_PK in tablespace INDX_TBS enforces primary constraints
of table ORAFRANCE.MATABLE in tablespace DATA_TBS
```

Le tablespace DATA_TBS peut être transporté contrairement à INDX_TBS. Effectivement, comment créer des indexes d'une table qui n'existe pas ?

Création d'une FK et test de transportabilité

```
SQL> create table orafrance.matable_fille
  2 (col1 NUMBER)
  3 tablespace INDX_TBS;

Table created.

SQL> alter table orafrance.matable_fille
  2 add constraint matable_matablefille_fk foreign key (col1)
  3 references orafrance.matable(col1);

Table altered.

SQL>
SQL> execute DBMS_TTS.TRANSPORT_SET_CHECK ('DATA_TBS',TRUE);

PL/SQL procedure successfully completed.

SQL> SELECT * FROM TRANSPORT_SET_VIOLATIONS;

no rows selected

SQL> execute DBMS_TTS.TRANSPORT_SET_CHECK ('INDX_TBS',TRUE);

PL/SQL procedure successfully completed.

SQL> SELECT * FROM TRANSPORT_SET_VIOLATIONS;

VIOLATIONS
--------------------------------------------------------------------------------
Constraint MATABLE_MATABLEFILLE_FK between table ORAFRANCE.MATABLE in
tablespace DATA_TBS and table ORAFRANCE.MATABLE_FILLE in tablespace INDX_TBS

Index ORAFRANCE.MATABLE_PK in tablespace INDX_TBS enforces primary constraints
of table ORAFRANCE.MATABLE in tablespace DATA_TBS
```

On a désormais un deuxième problème avec INDX_TBS, le tablespace contient la table MATABLE_FILLE liée à ORAFRANCE.MATABLE, ces deux tables étant dans deux tablespaces distincts.

Et si on vérifie le transport des deux tablespaces ?

```
SQL> execute DBMS_TTS.TRANSPORT_SET_CHECK ('DATA_TBS,INDX_TBS',TRUE);

PL/SQL procedure successfully completed.

SQL> SELECT * FROM TRANSPORT_SET_VIOLATIONS;

no rows selected
```

Ici, le transport des deux tablespaces est possible, toutes les contraintes étant levées.

Après avoir mis les tablespaces en READ ONLY on peut les transporter via les outils d'export et import d'Oracle.

En mode ligne de commande

```
exp userid = 'system/manager@dvp1'
                        TRANSPORT_TABLESPACE=y
                        TABLESPACES=DATA_TBS,INDX_TBS
                        TRIGGERS=y
                        CONSTRAINTS=y
                        GRANTS=y
                        ROWS=y

imp userid = 'system/manager@dvp2'
        TRANSPORT_TABLESPACE=y
        TABLESPACES=DATA_TBS,INDX_TBS
        DATAFILES=d:\dvp2\oradata\data_tbs.ora, d:\dvp2\oradata\indx_tbs.ora
```

Pour plus d'information sur les tablespaces transportables, je vous invite à lire la documentation Oracle : Transporting Tablespaces Between Databases.

1.2. Gestion de l'allocation d'espace

Lors de la création d'un segment (par exemple une table), Oracle crée un extent (dont la taille est définie dans les clauses de stockage) dans le tablespace cible de l'objet. Lorsqu'on rempli ce segment Oracle rempli les blocs de données qui constituent l'extent jusqu'à remplir l'extent entièrement et crée un nouvel extent si le précédent est plein.
Ainsi l'insertion de gros volumes peut générer un grand nombre de créations d'extents ce qui peut s'avérer coûteux à la longue.

Il existe 2 modes de gestion d'espace pour les tablespaces : la gestion par dictionnaire ou local (qui est apparue avec la version 8i).
Un tablespace géré par dictionnaire ordonne à Oracle de stocker les informations relatives à l'allocation d'espace dans le dictionnaire de données ce qui induit une charge supplémentaire pour toutes les opérations sur les objets d'un tablespace et oblige le DBA à tuner finement la taille des extents pour éviter une fragmentation excessive des fichiers et des accès intempestifs au dictionnaire de données.
Heureusement depuis la version 8i, Oracle s'affranchit de la notion d'extent grâce au tablespace géré localement (dit **locally managed**) qui stocke toutes les informations de stockage en entête du

tablespace.
Un tablespace est géré localement par défaut depuis la 9i et il est particulièrement conseillé de l'utiliser même avec la 8i. Effectivement, il permet d'éviter les contentions sur le dictionnaire de données, de simplifier la gestion de l'espace dans un tablespace qui devient complètement automatique et enfin, permet d'oublier la notion d'extent qui avait tendance à compliquer les choses inutilement. A noter que lors de la création de la base, le type du tablespace SYSTEM a un impact irréversible sur le type des autres tablespaces de la base : si le tablespace SYSTEM est LOCALLY MANAGED alors les autres tablespaces de la base devront aussi être géré localement.

Depuis la version 9i, il est possible d'allouer une taille de bloc différente de celui de la base pour chacun des tablespaces lors de sa création dès lors que le paramètre DB_nK_CACHE_SIZE ad hoc est renseigné :

```
CREATE TABLESPACE ora_data
        DATAFILE 'g:\oracle\oradata\orafrance\ORA_DATA01.dbf' size 100M
        BLOCKSIZE 4k;
```

A noter que les tablespaces d'une table ou d'un index partitionnés doivent tous avoir une même taille de bloc.

1.3. Le tablespace LOCALLY MANAGED et les extents

Il existe deux façons de gérer les extents en mode locally managed : UNIFORM SIZE et AUTOALLOCATE.
Le mode **UNIFORM SIZE** impose à Oracle de créer des extents de taille identique alors que le mode **AUTOALLOCATE** lui demande de créer des extents de plus en plus grands avec le nombre d'extents créés (les 16 premiers extents font 64k, les 64 suivant 1024k, etc.). A noter que ce dernier ne pourra pas être utilisé pour créer un tablespace temporaire.
Si le premier permet de suivre facilement l'allocation des extents, le deuxième permet au DBA de mieux gérer les systèmes qu'il ne maîtrise pas ou mal parce qu'il n'est pas maître des objets créés. Oracle précise d'ailleurs qu'il vaut mieux allouer de gros extents plutôt que de nombreux petits. De plus, il est intéressant de définir une taille d'extents proportionnelle à la valeur du paramètre DB_FILE_MULTIBLOCK_READ_COUNT.

Quoi qu'il en soit, il faudra éviter autant que possible de monopoliser des ressources système dans la gestion des extents.

1.4. La High-Water Mark ou niveau de flottaison

Voici une notion souvent ignorée mais je vais tacher d'y remédier ;-)
La High-Water Mark (ou niveau de flottaison) est un repère positionné par Oracle pour indiquer jusqu'à quel bloc d'un objet les données ont été renseignées pour savoir quand il doit s'arrêter lors d'un balayage de la table (FULL SCAN).
Par exemple, si dans une base avec une taille de bloc (DB_BLOCK_SIZE) de 16k on insère 40k de données dans une table, une HWM sera positionnée sur le 3° blocs de la table.
Sauf que la HWM ne redescend pas toute seule ! Voila qui pose un gros problème à Oracle...
Si l'on supprime beaucoup de lignes d'une table alors Oracle scannera quand même celle-ci jusqu'à la HWM au risque de balayer inutilement beaucoup de blocs vides.
Voila une des raisons qui oblige le DBA à réorganiser l'espace, puisque seul un TRUNCATE (et

évidemment une destruction :-)) de la table permet de remettre la HWM à 0. Pour reprendre l'exemple d'un excellent site qui traite du sujet, la HWM fonctionne comme le niveau du mercure dans un thermomètre : il ne redescend qu'en secouant le thermomètre ;-)

1.5. Paramètre de stockage et optimisation de l'espace

A l'insertion d'une ligne dans un segment, Oracle remplit le 1° bloc libre qu'il trouve en laissant PCTFREE % d'espace libre.
PCTFREE est la zone libre réservée aux mises à jour de données pour éviter la migration de celles-ci.
Effectivement, si un update fait en sorte qu'une ligne ne pourra pas être contenue dans un seul bloc, une partie de la ligne sera alors créée dans un autre bloc qui a peu de chance d'être contigus au premier.
Attention, il faut bien distinguer deux types de chaînage de données : la migration et le chaînage simple des données. Si le premier cas est décrit précédemment et peut être évité en changeant le PCTFREE, le deuxième cas lui est "normal" et relatif au cas où on insère des données dont la taille est supérieure à la taille du bloc.

Exemple

```
SQL> CREATE TABLE MATABLE (ID NUMBER NOT NULL,
  2  COMMENTS VARCHAR2(4000))
  3  TABLESPACE TOOLS PCTFREE 0 PCTUSED 0
  4  STORAGE ( INITIAL 4096 NEXT 4096 PCTINCREASE 0) ;

Table créée.

SQL> INSERT INTO matable (id) values (1);

1 ligne créée.

SQL> INSERT INTO matable (id) values (2);

1 ligne créée.

SQL> INSERT INTO matable (id) values (3);

1 ligne créée.

SQL> INSERT INTO matable (id) values (4);

1 ligne créée.

SQL> INSERT INTO matable (id) values (5);

1 ligne créée.

SQL> INSERT INTO matable (id) values (6);

1 ligne créée.

SQL> COMMIT;

Validation effectuée.

SQL> create table CHAINED_ROWS (
  2  owner_name         varchar2(30),
```

```
  3  table_name         varchar2(30),
  4  cluster_name       varchar2(30),
  5  partition_name     varchar2(30),
  6  subpartition_name  varchar2(30),
  7  head_rowid         rowid,
  8  analyze_timestamp  date );

Table créée.

SQL> analyze table matable list chained rows into chained_rows;

Table analysée.

SQL> select count(*) from chained_rows;

  COUNT(*)
----------
         0

SQL> declare
  1  toto varchar2(4000);
  2  begin
  3  for i in 1..4000 loop
  4  toto := toto || 'a';
  5  end loop;
  6  update matable set comments = toto;
  7  COMMIT;
  8  end;
  9  /

Procédure PL/SQL terminée avec succès.

SQL> analyze table matable list chained rows into chained_rows;

Table analysée.

SQL> select count(*) from chained_rows;

  COUNT(*)
----------
         2

SQL> declare
  2  toto varchar2(4000);
  3  begin
  4  for i in 1..4000 loop
  5  toto := toto || 'a';
  6  end loop;
  7  insert into matable values (7,toto);
  8  COMMIT;
  9* end;
  10 /

Procédure PL/SQL terminée avec succès.

SQL> truncate table chained_rows;

Table tronquée.

SQL> analyze table matable list chained rows into chained_rows;
```

```
Table analysée.

SQL>
SQL> select * from chained_rows;

aucune ligne sélectionnée
```

J'ai bien généré des lignes migrées dans le 1° cas (UPDATE) contrairement au 2° cas (INSERT).

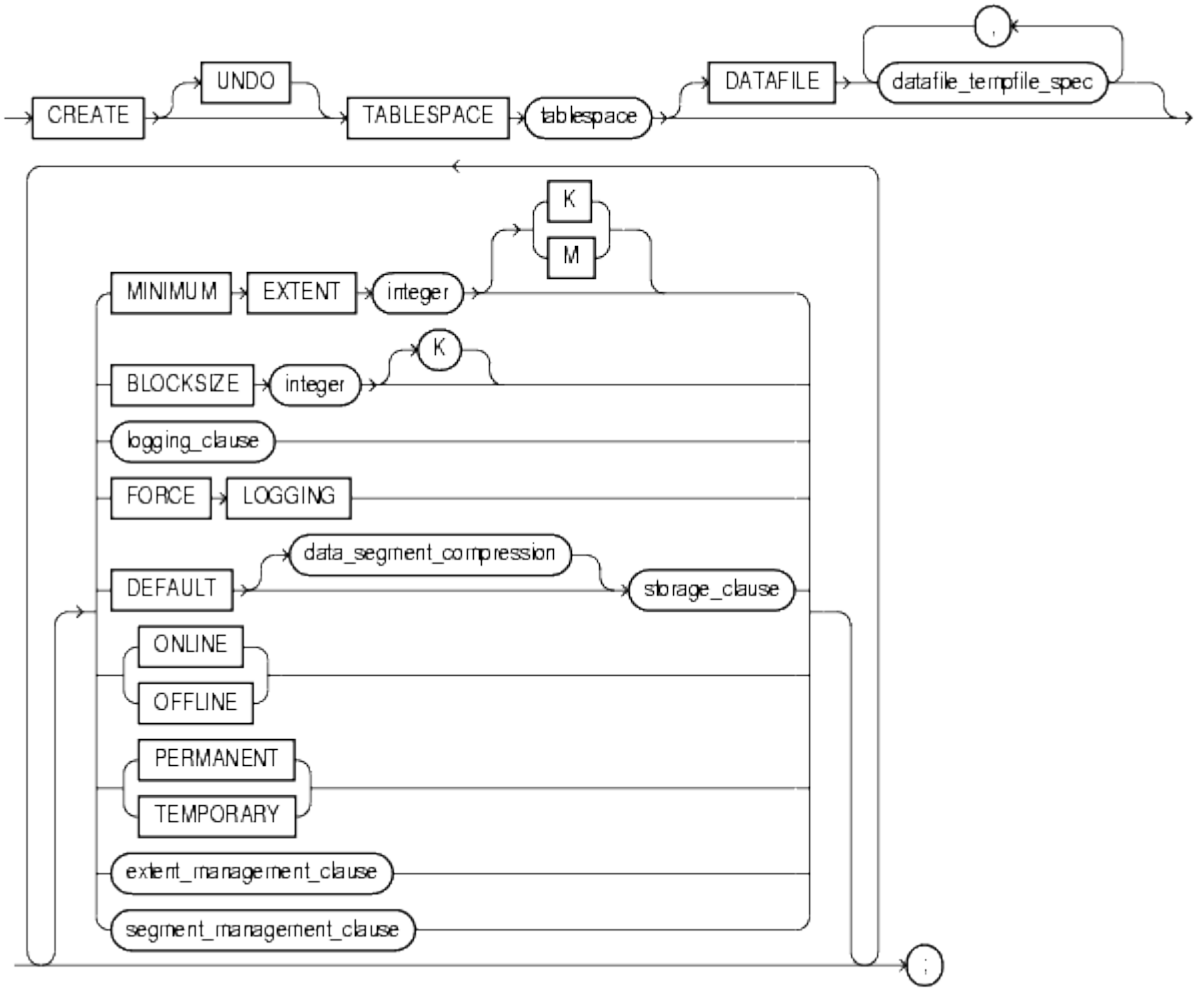
La FREELIST contient la liste des blocs dans lesquels Oracle peut écrire des données. Un bloc sort de la freelist dès qu'il est rempli à plus de PCTUSED %. Il y retourne dès que le niveau redescend en dessous de PCTUSED : en cas de DELETE de lignes ou d'UPDATE avec des valeurs plus petites.

Donc, si le PCTFREE est trop grand, on réserve trop d'espace aux UPDATE et on perd de l'espace. S'il est trop petit on augmente le risque de lignes migrées. Et si le PCTUSED est trop grand, Oracle perd beaucoup de temps dans la gestion de la freelist mais si il est trop petit l'espace disque est une nouvelle fois perdu puisque le bloc n'est pas considéré comme libre pour l'écriture de données.

2. Syntaxe commentée de création d'un tablespace

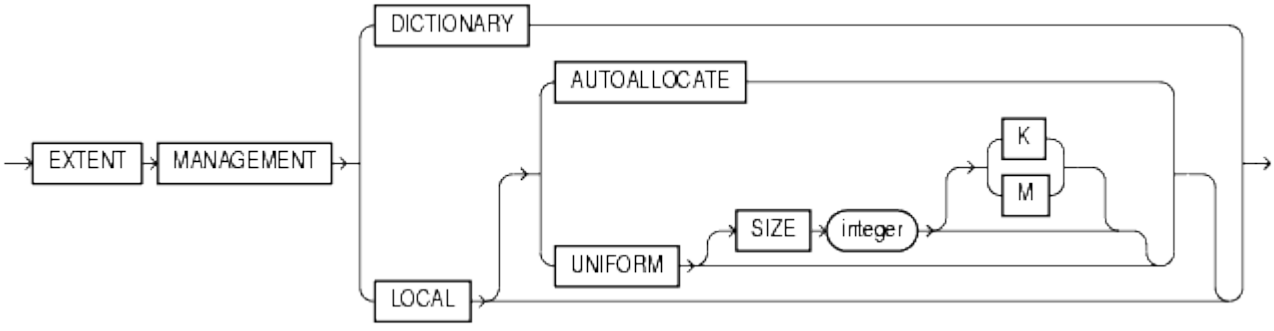
2.1. Création de tablespace

Voici la structure complète d'un ordre CREATE TABLESPACE :



LOGGING: permet d'indiquer si la création d'objet dans le tablespace doit être inscrite dans les redo logs.
FORCE LOGGING: permet de forcer le LOGGING même sur les objets ayant l'option NOLOGGING.
Cette option est invalide pour les tablespaces de type TEMPORARY ou UNDO.
ONLINE ou OFFLINE: permet de rendre le tablespace disponible ou non, juste après la création.
TEMPORARY: permet de créer un tablespace temporaire géré par le dictionnaire.
Pour créer un tablespace temporaire géré localement, il faut utiliser la commande CREATE TEMPORARY TABLESPACE.
data_segment_compression: peut prendre les valeurs COMPRESS ou UNCOMPRESS selon que l'on veut compresser ou non les données.

extent_management_clause :



LOCAL ou DICTIONARY : permet d'indiquer que le tablespace est géré localement (cf. 1.3) ou par dictionnaire.
UNIFORM ou AUTOALLOCATE : permet d'indiquer que le tablespace s'agrandit de manière uniforme (exprimé en taille d'extents) ou automatiquement.

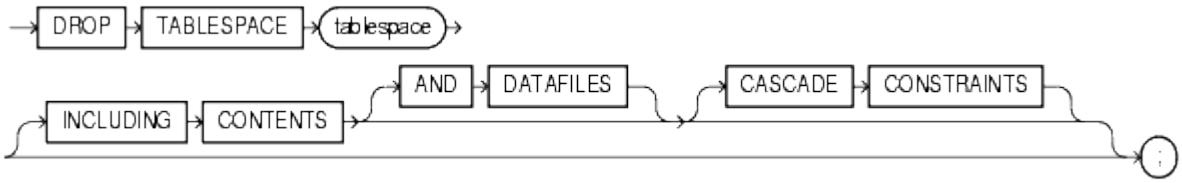
segment_management_clause :

AUTO : permet de laisser la base gérer l'espace libre. Oracle ignore alors les paramètres PCTUSED, FREELIST et FREELIST GROUPS des objets du tablespace.

Attention : si le tablespace est en LOCAL UNIFORM, alors il faut s'assurer que le prochain extent pourra contenir au moins 5 blocs, c'est à dire que la taille définie pour les prochains extents est d'au moins 6 fois la taille du bloc de base de données.
Si au contraire, le tablespace est en LOCAL AUTOALLOCATE et que les blocs de base sont au moins de 16k, alors Oracle créera des extents d'au moins 1 Mo.

2.2. Suppression de tablespace

Voici la structure complète d'un ordre DROP TABLESPACE :

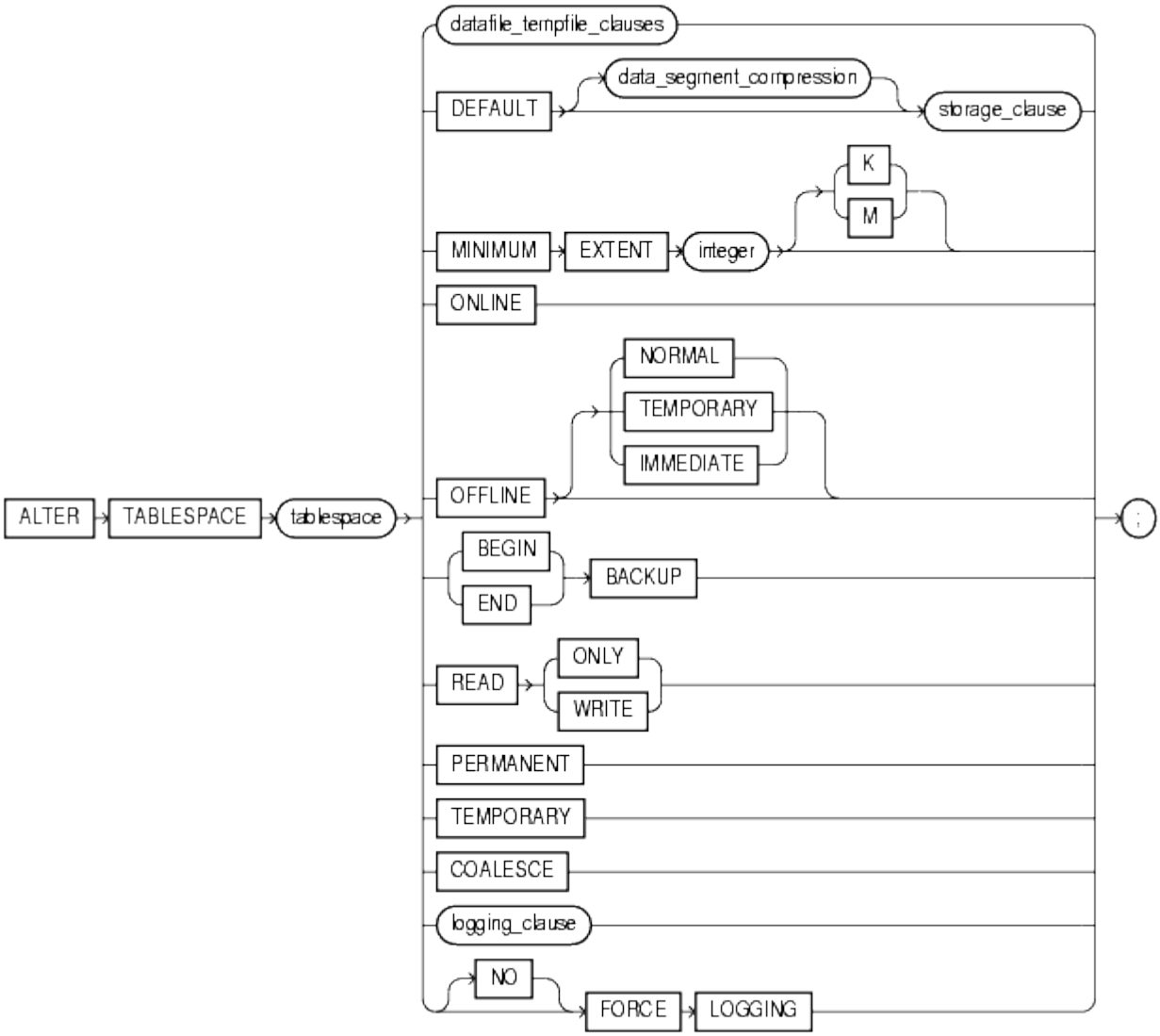


INCLUDING CONTENTS : permet de supprimer le contenu du tablespace. Si cette option n'est pas précisée, il sera impossible de supprimmer un tablespace contenant des objets.

Attention : Les tablespaces contenant des tables partitionnées ne pourront pas être supprimés même si cette option est spécifié.
AND DATAFILES: permet de supprimer également les fichiers, une trace est alors inscrite dans le fichier des alertes pour chacun des fichiers supprimés.
CASCADE CONSTRAINTS: permet de supprimer les contraintes d'intégrité référentielles vers des objets d'autres tablespaces et les contraintes unique du tablespace supprimé.

2.3. Modification de tablespace

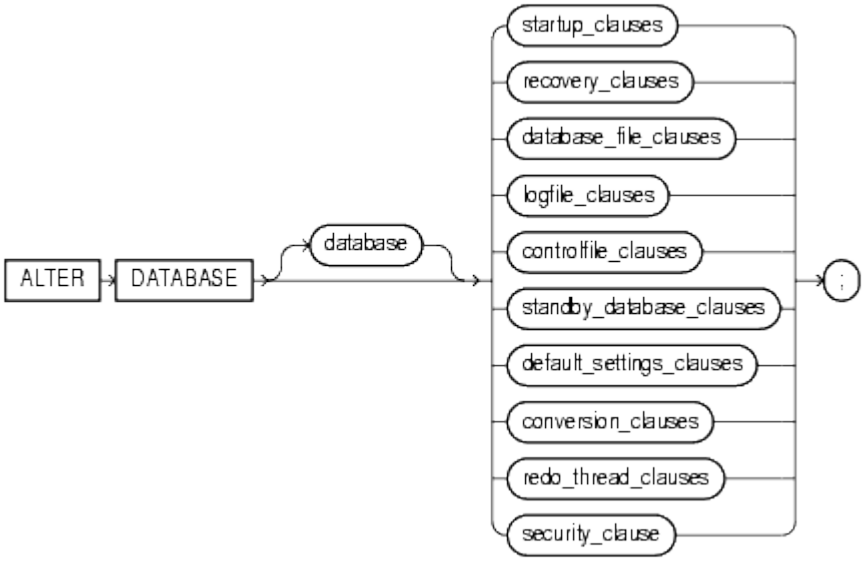
Voici la structure complète d'un ordre ALTER TABLESPACE :



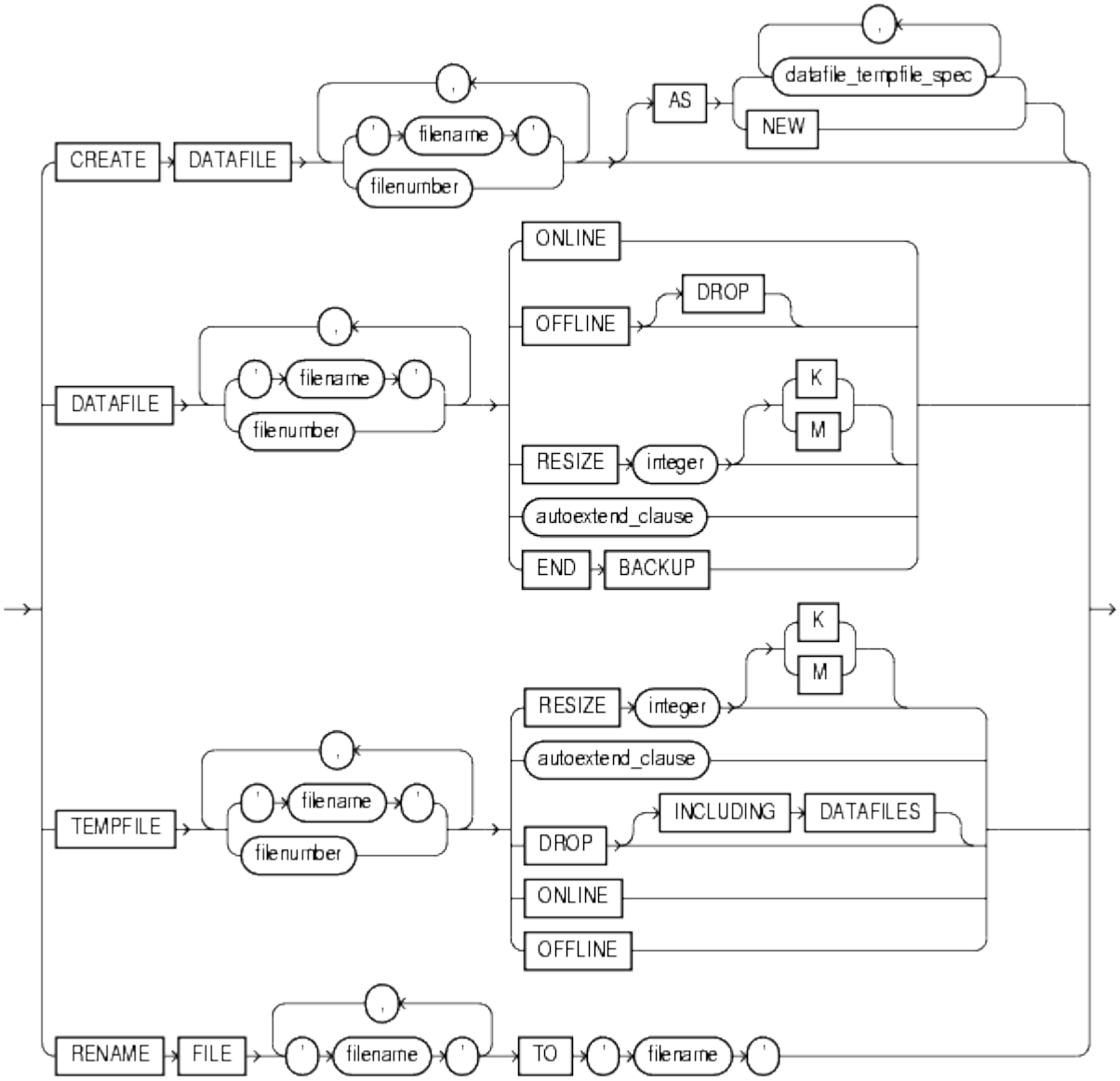
COALESCE : permet de fusionner les extents libres contigus.
BEGIN ou END BACKUP : permet d'indiquer le début ou la fin d'une sauvegarde base ouverte de la base. Cette option n'est pas applicable aux tablespaces temporaires ou en READ ONLY

2.4. Modification de la base de données (spécifique au tablespace)

Voici la structure complète d'un ordre ALTER DATABASE :



Voici la structure de la partie database_file_clauses spécifique à notre sujet :



3. Exemples d'ordres SQL pour la gestion des tablespaces

Création d'un tablespace

```
CREATE TABLESPACE ora_data
DATAFILE 'g:\oracle\oradata\orafrance\ORA_DATA01.dbf' size 100M,
         'g:\oracle\oradata\orafrance\ORA_DATA02.dbf' size 100M
MINIMUM EXTENT 500K --(uniquement V8)
DEFAULT STORAGE (initial 500K next 500K MAXEXTENTS 500 PCTINCREASE 0);
```

Création d'un tablespace géré localement

```
CREATE TABLESPACE ora_data
   DATAFILE 'g:\oracle\oradata\orafrance\ORA_DATA01.dbf' SIZE 10M
   EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K;
```

Si la taille d'un bloc est de 2k alors ce tablespace s'étendra de 64 blocs à chaque nouvel extent.

Placer un tablespace à l'état OFFLINE

```
ALTER TABLESPACE ora_data OFFLINE;
```

Cette commande peut être utile pour éviter que les données du tablespace soient accessibles, particulièrement lors des opérations de maintenance.

Création d'un tablespace UNDO de 10 Mo autoextensible

```
CREATE UNDO TABLESPACE undotbs
   DATAFILE 'undotbs.dbf'
   SIZE 10M AUTOEXTEND ON;
```

Suppression d'un tablespace et ses contraintes

```
DROP TABLESPACE ora_data INCLUDING CONTENTS CASCADE CONSTRAINTS;
```

Modifier les clauses de stockage d'un tablespace

```
ALTER TABLESPACE ora_data
DEFAULT STORAGE ( INITIAL 100K
                  NEXT 100K
                   MINEXTENTS 1
                   MAXEXTENTS 300
                  PCTINCREASE 1);
```

Ajout d'un datafile

```
ALTER TABLESPACE ora_data ADD
DATAFILE 'g:\oracle\oradata\orafrance\ORA_DATA03.dbf' size 100M;
```

Déplacement d'un datafile

```
ALTER TABLESPACE ora_data OFFLINE NORMAL;
-- Après avoir copié le fichier ORA_DATA03.dbf
-- de g:\oracle\oradata\orafrance vers f:\oracle\oradata\orafrance
```

```
ALTER TABLESPACE ora_data
  RENAME DATAFILE 'g:\oracle\oradata\orafrance\ORA_DATA03.dbf'
  TO              'f:\oracle\oradata\orafrance\ORA_DATA03.dbf';
ALTER TABLESPACE ora_data ONLINE NORMAL;
```

Suppression d'un datafile

```
ALTER DATABASE
        DATAFILE 'g:\oracle\oradata\orafrance\ORA_DATA03.dbf' OFFLINE DROP;
```

Modifie la taille d'un datafile

```
ALTER DATABASE
DATAFILE 'g:\oracle\oradata\orafrance\ORA_DATA03.dbf' resize 50M;
```

Suivi du remplissage des tablespaces

```
set lines 150
col Max_Libre for 999,999,999,999
col Total_Dispo for 999,999,999,999
col Taille_Ts for 999,999,999,999
col Ts_Frag_Ext_Libres for A50
col tablespace_name for A10

SELECT tablespace_name, Taille_Ts, Max_Libre, Total_Dispo,
           to_char(100*Total_Dispo/Taille_Ts, '999.99') || '%' AS pct_dispo,
DECODE(Total_Dispo - Max_Libre,0,'Tous les extents libres sont contigus',
       'Le nb d''extents libres non contigus est de: ' || Nb_Frag)
FROM ( SELECT tablespace_name,
       sum(bytes) AS Taille_Ts
       FROM dba_data_files
       GROUP BY tablespace_name
      ),
        ( SELECT tablespace_name AS fs_ts_name,
                        max(bytes) AS Max_Libre,
                        sum(bytes) AS Total_Dispo,
                        count(*) AS Nb_Frag
                FROM dba_free_space
                GROUP BY tablespace_name
        )
WHERE tablespace_name = fs_ts_name
ORDER BY 5 desc;
```

4. Tordons le cou aux mythes

4.1. Trop d'extents dégrade les performances

Voila bien une croyance défendue par de nombreux DBA sans qu'aucune explication ne puisse être fournie. Elle consiste donc à dire que lorsque le nombre d'extents d'un objet est trop important, il faut le reconstruire pour remplacer n petits extents par un initial extent énorme.
Tout d'abord, la question est immédiatement réglée pour les tablespaces gérés localement qui, comme on l'a vu dans le paragraphe 1.1, permettent de s'affranchir complètement de la notion d'extent.

Mais quid des afficionados des tablespaces gérés dans le dictionnaire ?
Je suppose que ce mythe repose sur une croyance : les données d'un extent sont organisées sur le disque de telles sortes qu'elles occupent des blocs contigus.
Mais pensez-vous vraiment que c'est possible ? Quel chance a-t-on de retrouver sur le disque suffisament de blocs contigus pour y placer la totalité du gros extent ? Et si d'aventure il y avait effectivement suffisament de blocs contigus, il resterait quand même une problématique propre au fonctionnement d'un disque.
Heureusement pour nous, un disque n'est pas dévoué à une tâche seulement, de telle sorte qu'il lit et écrit des données quasiment simultanément de telle sorte que les têtes de lecture des disques se déplacent fréquement. Si une lecture intervient pendant l'écriture de votre gros extent initial alors le disque, après sa lecture, continuera l'écriture dans le bloc libre le plus proche du dernier bloc lu (pour éviter que la tête de lecture ne parcoure un trajet trop important), et bien entendu, on n'a quasiment aucune chance que ce bloc libre soit contigü au bloc écrit précédemment.
Ajouter à cela le fonctionnement en cluster ou dans une baie de disque et cela finira je l'espère de vous convaincre.

Il est temps pour moi de pondérer ces propos. En effet, la parole d'Oracle étant parfois idyllique, force est de constater qu'un excés vraiment important d'extents pose problème. Ainsi, on essayera de réduire le nombre d'extent sans verser dans la paranoïa.

4.2. Un tablespace de données et un tablespaces d'index suffisent amplement

Là, Oracle propose des chiffres très clair, un objet a une gestion d'espace optimale lorsqu'il contient jusqu'à 1024 extents. Ainsi, on fera des tablespaces pour les petits, moyens et gros objets.

Taille d'extent selon la taille des objets

- Juqu'à 160 Mo, INITIAL=NEXT=160 ko
- De 160 Mo à 5 Go, INITIAL=NEXT=5 Mo
- Au delà de 5 Go, INITIAL=NEXT=160 Mo

Ainsi, une table de 160M pourra bien contenir jusqu'à 1024 extents de 160 ko.