Administration
Le stockage des données

Anne-Cécile Caron

Master MIAGE - BDA

1er trimestre 2010-2011

Mémoires

Le SGBD manipule des données, présentes dans plusieurs types de mémoires :

- Mémoire primaire : mémoire cache, mémoire centrale (RAM)
  Une donnée doit être en mémoire primaire pour être manipulée par le SGBD (construire le résultat d'une requête)
- Mémoire secondaire : disque magnétique
  Support qui permet la persistance des données. Le SGBD doit lire ou écrire sur la mémoire secondaire. L'unité de lecture/écriture est le bloc.
- Mémoire tertiaire : bande, disque optique (DVD).
  Support utilisé pour des sauvegardes.

Mémoires

- La mémoire secondaire peut être une ressource critique : plusieurs accès concurrents au même bloc
- La façon dont les données sont réparties dans la mémoire secondaire influence les performances
- Il y a 2 solutions : la répartition (striping) sur plusieurs disques, et la redondance (mirroring)
- disques RAID = *redundant arrays of independent disks*. Peut gérer la redondance, la répartition ou les deux.

Gestion de fichiers

- Un SGBD utilise des fichiers OS pour stocker ses données
- MAIS il gère lui-même l'espace à l'intérieur de ces fichiers :
  - taille de fichier supérieure à celle supportée par un OS
  - portabilité : gestion de la mémoire indépendamment de l'OS
  - gestion des blocs vides, allocation de blocs
  - lecture anticipée de blocs (prefetching) en cas de requête.

Instance et base de donnée Oracle

- Une base de données est définie par un nom (par exemple `filora`)
- Une base contient un certain nombre de fichiers
- Quand une base est démarrée sur le serveur, celui-ci lui a associé une instance Oracle, c'est-à-dire :
  - une zone d'échanges appelée `System Global Area` ou database cache.
  - un ensemble de processus systèmes.
  - Une zone mémoire pour les processus serveurs, la `Program Global Area`.

Les fichiers

Il existe plusieurs types de fichiers

- Data files : contiennent toutes les données de la base, toutes les structures logiques ou physiques (tables, indexes, rollback segment).
- Redo Log files : contiennent des informations sur les données modifiées. Ils sont utilisés en cas de perte des data files.
- Control files : ils contiennent la description physique de la base, des informations sur les fichiers et la cohérence de la base (checkpoint). Ils sont utilisés au démarrage d'une instance, et pour la restauration. Ils sont modifiés à chaque modification structurelle de la base.
- fichiers d'archivage des Redo log files (optionnel)
- fichiers de paramètres (fichier texte, ou depuis la version 10, fichier SPFILE, binaire géré par le serveur).
- fichiers de trace (utilisés par les processus du SGBD)
- fichiers de sauvegarde

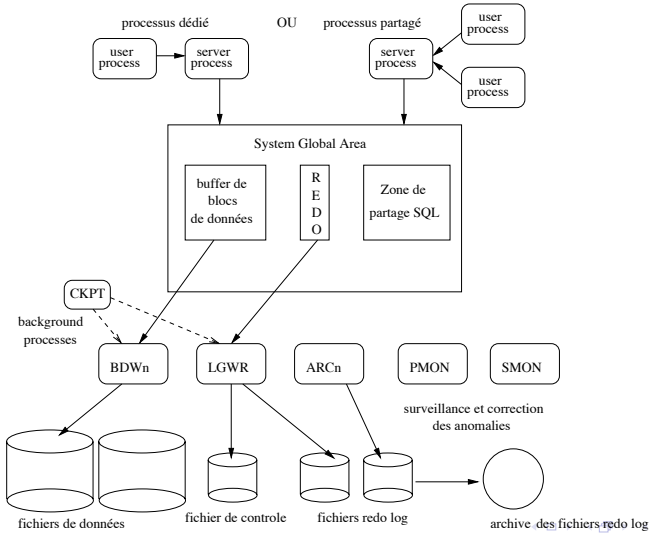
System Global Area

- la SGA rassemble des informations partagées par les différents utilisateurs.
- L'espace mémoire nécessaire à la SGA est
  - alloué au démarrage d'une instance
  - restitué à la fermeture de cette instance.
- Les données de la SGA sont structurées en plusieurs buffers
  - Les buffers de données (data block buffers)
  - Les buffers de journalisation (redo log buffers)
  - La zone de partage des ordres SQL, utilisée pour mémoriser, analyser, traiter les ordres SQL des utilisateurs.

Les processus

- User Process : processus créé et maintenu pour exécuter le code d'un programme applicatif (par exemple, SQLPlus). Il y a 1 user process par session utilisateur.
- System Process :
  - Server Process : assurent la communication entre la SGA et les User Process. Chaque processus serveur dispose d'une zone mémoire non partagée. Ces zones sont regroupées dans la `Program Global Area`. Il dispose en particulier d'une zone SQL privée.
    - Si la session est connectée à un serveur "dédié", alors les zones SQL propres à chaque processus serveur sont dans la PGA.
    - Si la session est connectée à un serveur "partagé" alors les zones SQL propres à chaque processus serveur sont partagées dans la SGA.
  - Background Process : les mécanismes internes de gestion des données, des journaux ...

Les processus (2)



Les processus (3)

Sur la figure précédente, on y voit différents background process :

- DBWn : écrit les blocs modifiés de la base dans les fichiers de données. *il y a plusieurs processus de ce type DBW0, DBW1, ···*
- LGWR : écrit le contenu du buffer REDO LOG de la SGA dans le fichier REDO LOG lors d'un commit.
- ARCn : recopie les fichiers REDO LOG pleins sur un fichier archive (optionnel). Il y a plusieurs processus de ce type.
- CKPT : responsable des checkpoints, i.e. écriture sur disque des données et logs.
- SMON : rétablit la cohérence du système après un incident, et libère les ressources utilisées par le système.
- PMON : récupère les anomalies des processus utilisateurs (supprime les process en erreur, libère les ressources utilisées par ces process ...)

exemple d'exécution

- 1. Une machine cliente (user process) se connecte sur le serveur
- 2. Le serveur détecte la demande de connexion et créé un processus serveur *PS* dédié à ce client
- 3. L'utilisateur exécute un update et valide la transaction (commit)
- 4. *PS* reçoit la requête SQL, regarde dans la zone de partage des ordres SQL s'il y a une requête similaire. Si oui, *PS* vérifie les droits du client et exécute la requête, si non, on alloue de l'espace à cette nouvelle requête afin de l'analyser et de la traiter.

exemple d'exécution

- 5. *PS* retrouve toutes les données nécessaires à la requête, soit dans la SGA, soit dans les fichiers.
- 6. *PS* exécute la modification (update) dans la SGA. Le process DBWn écrit éventuellement les blocs modifiés sur le disque. Le process LGWR enregistre tout de suite la transaction dans les fichiers redo log car il y a eu un commit.
- 7. Si la transaction s'est bien déroulée, *PS* envoie un message à l'application cliente. Sinon, il envoie un message d'erreur.
- 8. Pendant tout ce temps, les processus background fonctionnent, et il peut y avoir d'autres processus clients donc d'autres processus serveurs.

Structure logique de la base

- Au niveau système, la base de données est un ensemble de fichiers (= une suite de blocs en mémoire secondaire)
- Pour le SGBD, on abstrait ces fichiers en structurant les données dans des TABLESPACES
- 1 tablespace est associé à 1 ou plusieurs (< 256) fichiers de données.

Tablespaces

Il existe différents types de tablespaces :

- Tablespace permanent : pour stocker les données (tables, index). 1 objet correspond à 1 *segment*. On trouvera donc des segments de données (tables) et des segments d'index.
- Tablespace temporaire : tablespace spécifique aux opérations de tri, pour lesquelles la mémoire primaire ne suffit pas (`SORT_AREA_SIZE`). Il est réservé au système, il ne reçoit pas d'objet de la base de données.
- Undo Tablespace : réservé à l'annulation des commandes (cf cours sur les transctions).

Exemple : la base du FIL

```
-- créés à l'installation du serveur Oracle :
SYSTEM -- contient le dictionnaire des données
UNDOTBS1 -- Pour les UNDO
SYSAUX -- TS système auxiliaire, utilisé par des produits Oracle
TEMP -- pour les segments temporaires
USERS -- pour les utilisateurs

-- créés par les enseignants :
LICENCE  MIAGE3  BDAMIAGE  IAGL ...
```


Tablespace et utilisateurs

Généralement, quand on crée un utilisateur, on lui affecte un tablespace par défaut.
Exemple :

```
create user Dupont identified by toto
default tablespace BDAMIAGE
temporary tablespace TEMP ;

-- pas de quota sur aucun tablespace
grant unlimited tablespace to Dupont;
-- si on veut fixer des quotas :
alter user Dupont
    QUOTA 10M ON BDAMIAGE
    QUOTA 5M ON USERS;
```

Tablespace

Un tablespace contient des segments, chaque segment est constitué d'*extents*.
Sous Oracle 10, il y a deux façons de gérer les tablespaces, et donc les extents :

- avec le dictionnaire : la gestion de l'espace se fait en utilisant le dictionnaire des données. Dans ce cas, on peut préciser à la création du tablespace les paramètres de stockage de ces segments.
- localement : les informations permettant de gérer l'espace sont stockées dans le tablespace lui-même.
  - pas de contention de mise-à-jour du dictionnaire
  - fusion des espaces libres contigus pour optimiser l'espace libre

Par défaut, pour les tablespace autres que `SYSTEM`, la gestion se fait localement.

Tablespace et objets

- Par défaut, une table est créée dans le tablespace par défaut de son propriétaire.
- de même un index sur une table est créé dans le même tablespace que cette table
- on peut définir explicitement le tablespace pour une table ou un index :

```
create table T (a NUMBER, b NUMBER )
tablespace TABLES_TS;

alter table T add
(constraint t_pk primary key (a)
using index tablespace INDEX_TS );
```

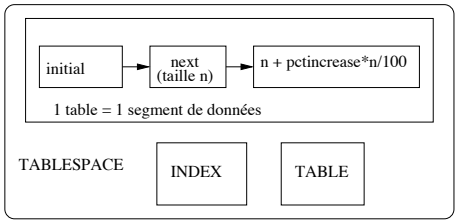

Gestion par le dictionnaire

Un segment est constitué d'extents, dont la taille est définie par les paramètres de stockage.
Pour un tablespace géré via le dictionnaire, on peut définir ces paramètres à la création du tablespace

- `initial` taille du premier extent
- `next` taille du second extend
- `pctincrease` pourcentage d'augmentation de la taille de chaque nouvel extent.
- `minextents` nombre minimun d'extents par segment (donc nombre d'extents à la création du segment)
- `maxextents` nombre maximum d'extents par segment
- `freelists` nombre de listes de blocs libres qu'il faut gérer (si *n* listes, on peut faire *n* insertions en parallèle).

```
CREATE TABLESPACE ESSAI
DATAFILE 'C:\orant\database\file1.ora' SIZE 10M
DEFAULT STORAGE ( INITIAL 1024K NEXT 1024K PCTINCREASE 0);
```


Structure d'un segment



Gestion locale

Pour un tablespace géré localement, il existe une table d'allocation représenté par un *bitmap* stocké dans le tablespace. Chaque bit indique si l'extent correspondant est libre ou non. Il existe 2 façons d'allouer des extents :

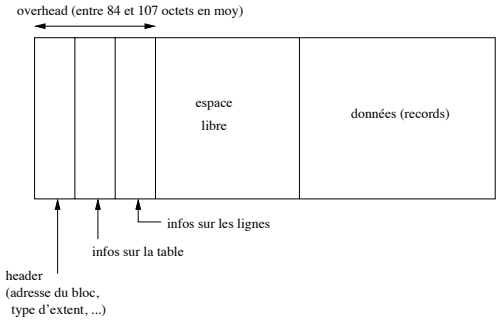
- UNIFORM : tous les extents ont la même taille
- AUTOALLOCATE : les extents sont de plus en plus grands.

Dans ce cas les paramètres de stockage next, pctincrease, minextents, maxextents n'ont pas de sens (il n'y a pas de clause `DEFAULT STORAGE`).

blocs

- Un extent est constitué de blocs (data blocks) qui peuvent être configurés grâce aux paramètres suivants :
  - `pctfree` pourcentage du bloc qui ne doit pas être rempli quand on insère des lignes dans la table (réserver cet espace pour les update)
  - `pctused` pourcentage du bloc qui doit être à nouveau rempli avant d'insérer des lignes (après avoir atteint un espace libre $\leq$ `pctfree`).
- La taille d'un bloc dépend de l'OS.
- Pour les segments de données et d'indexes, une liste des blocs libres est gérée (blocs dans lesquels on peut insérer car espace libre > PCTFREE)

gestion de l'espace d'un bloc



Supposons que `pctfree = 20` et `pctused = 40`.

- Tant que l'espace libre du bloc est supérieur à 20% de l'espace total, on peut insérer des nouvelles lignes.
- Ensuite, Tant que l'espace occupé n'est pas redescendu à 40% de l'espace total, on ne peut faire que des modifications et suppressions.
- et ainsi de suite ...

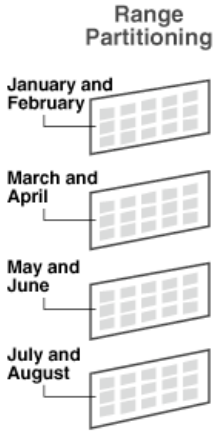
Partitionnement d'une table ou d'un index

- Lorsqu'une table est volumineuse, on peut la partitionner, i.e. la répartir sur plusieurs tables plus petites (1 partition occupe 1 bloc).
- Le découpage logique de la table permet un accès plus rapide aux informations (moins de lectures disques)
- C'est transparent pour l'utilisateur (requête SQL sur une table, pas besoin de savoir qu'elle est partitionnée)
- La partition se fait selon une *clé* de partition : c'est un sous-ensemble des colonnes qui permet à chaque ligne de la table d'être affectée sans ambiguité à 1 partition.
- Toutes les partitions ont le même schéma logique (ensemble de colonnes) mais peuvent avoir des attributs physiques différents (comme les `TABLESPACE`s).

Méthodes de partitionnement

- Une table peut être partitionnée selon 4 méthodes :
  1. Range : en fonction d'intervalles de valeurs
  2. List : en fonction de listes de valeurs
  3. Hash : selon une fonction de hashage
  4. Composite : utilise la méthode *range* et fabrique pour chaque partition une sous-partition en utilisant la méthode *List* ou *Hash*
- Les indexes, comme les tables, peuvent être partitionnés. Ils peuvent être partitionnés
  - en même temps que la table (index local), dans ce cas 1 partition d'index pour 1 partition de la table
  - ou indépendamment du partitionnement de la table (index global)

Range - exemple

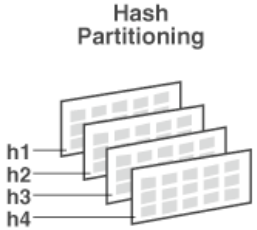



```
CREATE TABLE sales_range
(salesman_id  NUMBER(5),
salesman_name VARCHAR2(30),
sales_amount  NUMBER(10),
sales_date    DATE)
PARTITION BY RANGE(sales_date)
(
PARTITION sales_jan_feb2000
VALUES LESS THAN(TO_DATE('03/01/2000',
                         'MM/DD/YYYY')),
PARTITION sales_mar_apr2000
VALUES LESS THAN(TO_DATE('05/01/2000',
                         'MM/DD/YYYY')),
PARTITION sales_may_jun2000
VALUES LESS THAN(TO_DATE('07/01/2000',
                         'MM/DD/YYYY')),
PARTITION sales_jul_aug2000
VALUES LESS THAN(TO_DATE('09/01/2000',
                         'MM/DD/YYYY'))
);
```


List - exemple


```
CREATE TABLE sales_list
(salesman_id  NUMBER(5),
salesman_name VARCHAR2(30),
sales_state   VARCHAR2(20),
sales_amount  NUMBER(10),
sales_date    DATE)
PARTITION BY LIST(sales_state)
(
PARTITION sales_east
VALUES ('New York', 'Virginia', 'Florida'),
PARTITION sales_west
VALUES('California', 'Oregon','Hawaii'),
PARTITION sales_central
VALUES('Texas', 'Illinois','Missouri')
);
```


Hash - exemple




```
CREATE TABLE sales_hash
(salesman_id  NUMBER(5),
salesman_name VARCHAR2(30),
sales_amount  NUMBER(10),
week_no       NUMBER(2))
PARTITION BY HASH(salesman_id)
PARTITIONS 4
STORE IN (ts1, ts2, ts3, ts4);
```

La clause STORE permet de répartir les éléments de la partitions dans différents tablespaces (ts1,ts2,...).

Composite - exemple Range-Hash

```
CREATE TABLE sales_composite
(salesman_id  NUMBER(5),
 salesman_name VARCHAR2(30),
 sales_amount  NUMBER(10),
 sales_date    DATE)
PARTITION BY RANGE(sales_date)
SUBPARTITION BY HASH(salesman_id)
SUBPARTITION TEMPLATE(
SUBPARTITION sp1 TABLESPACE ts1,
SUBPARTITION sp2 TABLESPACE ts2,
SUBPARTITION sp3 TABLESPACE ts3,
SUBPARTITION sp4 TABLESPACE ts4)
(PARTITION sales_jan2000 VALUES LESS THAN(TO_DATE('02/01/2000','MM/DD/YYYY'))
 PARTITION sales_feb2000 VALUES LESS THAN(TO_DATE('03/01/2000','MM/DD/YYYY'))
 PARTITION sales_mar2000 VALUES LESS THAN(TO_DATE('04/01/2000','MM/DD/YYYY'))
 PARTITION sales_apr2000 VALUES LESS THAN(TO_DATE('05/01/2000','MM/DD/YYYY'))
 PARTITION sales_may2000 VALUES LESS THAN(TO_DATE('06/01/2000','MM/DD/YYYY')));
```


Gestion de la mémoire – conclusion

L'administrateur peut

- Utiliser la réplication/répartition sur des disques RAID, pour optimiser les I/O
- Gérer les tablespaces :
  - Créer des tablespace et affecter les objets à ces tablespaces, en séparant par exemple les index et les tables.
  - La gestion locale est plus performante que par le dictionnaire.
  - modifier les paramètres de stockage : il vaut mieux créer des gros extents plutôt que de nombreux petits (pour 1 objet, gestion d'espace optimale jusqu'à 1024 extents).
- Pour des tables volumineuses, envisager un partitionnement, et stocker les partitions dans des tablespaces différents.
- désallouer de l'espace inutilisé dans un segment de donnée ou d'index, effectuer des défragmentations
- vérifier/réparer les blocs d'un fichier de données