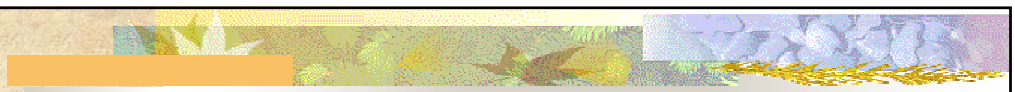
Conception physique de bases de données

J. Akoka / I. Wattiau



1. Introduction

- Objectif : Obtenir les meilleures performances possibles en termes de :
  - temps d'accès
  - espace mémoire utilisé
  - espace disque occupé.

  * Minimiser le temps d'insertion et de mise à jour des données dans la base
  * Minimiser le temps d'accès aux données
  * Minimiser la place occupée par les données en mémoire secondaire

Dans la plupart des cas :

* L'architecture du SGBD repose sur le Système de Gestion de Fichiers (SGF) de la machine
- Avec éventuellement des techniques de stockage et d'accès supplémentaires.

L'obtention de bonnes performances suppose :

- d'opérer un «bon» découpage de la base de données en fichiers, c'est l'objet de la conception physique,
- d'effectuer une gestion de fichiers efficace, c'est le rôle du système de gestion de base de données et du système de gestion de fichiers sous-jacent.

2. Supports de mémoire secondaire

* *Le disque magnétique*
  - support de stockage à haut débit
  - permet un accès direct aux données
  - c'est le support privilégié de tous les systèmes informatiques

* *La bande magnétique*
  - support le moins onéreux et le moins fragile
  - faible débit
  - organisation purement séquentielle
  - utilisée pour les sauvegardes.

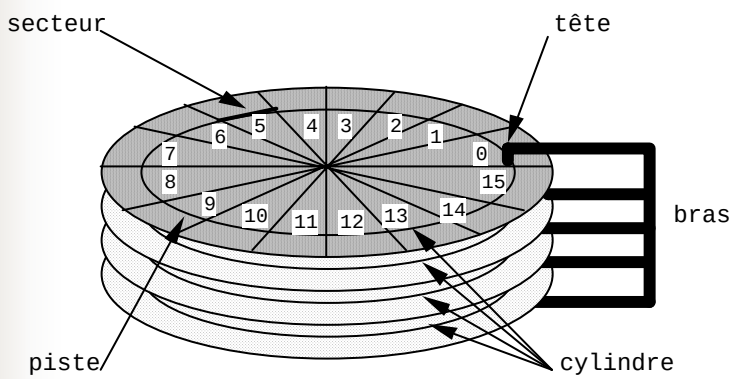
Les disques magnétiques

* Plateau magnétisé sur les deux faces
* Tourne à vitesse constante
* Composé de pistes concentriques (200 à 400 pistes par disque)
* Chaque piste est découpée en secteurs
* Un volume est un ensemble de disques superposés et séparés par un espace (pour le passage des têtes de lecture/écriture)
* *Disque à tête fixe :* une tête de L/E par piste
* *Disque à tête mobile :* les têtes de L/E sont reliées à un bras qui se déplace pour se positionner sur la piste choisie
* *Cylindre* : ensemble des pistes placées dans un même plan vertical

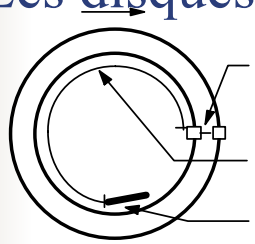
Le temps d'accès (- 10 ms) pour un disque à tête fixe : environ 50000 fois le temps d'accès en mémoire centrale

=> Nécessité d'optimiser les E/S

Les disques magnétiques (suite)



Les disques magnétiques (suite)



b	r	d	T = 1 K	T = 20 K
5 ms	6 ms	5 Mo/s	12 ms	15 ms
25 ms	8,3 ms	250 Ko/s	35 ms	115 ms

Stratégies d'E/S disques

* Il y a asynchronisme entre le traitement par le processeur central et le déroulement des E/S

=> le module de gestion des disques gère une file de requêtes d'E/S

=> on peut optimiser le temps de réponse global des requêtes en :

- minimisant les déplacements de bras pour les unités à têtes mobiles,
- minimisant le nombre de rotations des disques,

mais il ne faut pas choisir un algorithme très complexe !

Exemples de stratégies

SLTF (Shortest Latency Time First) :

on exécute à chaque fois la requête dont les secteurs sont le plus près des têtes de Lecture/Ecriture

C'est la meilleure stratégie pour la gestion des disques à têtes fixes.

SSTF (Shortest Seek Time First) :

on exécute en premier la requête la plus proche des têtes de L/E (pour les disques à têtes mobiles).

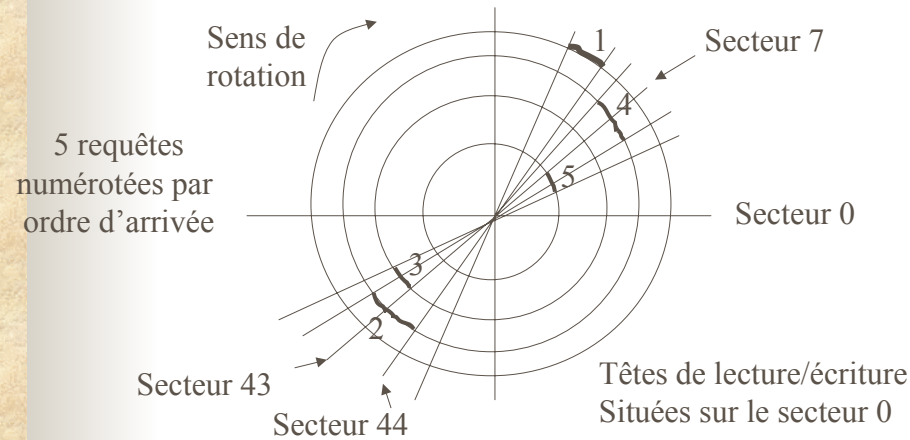
SCAN :

les têtes de L/E font la navette entre le premier et le dernier cylindre du disque et traitent au passage les requêtes concernant le cylindre situé sous les têtes de L/E.

SCAN est meilleure que SSTF dès qu'il y a un grand nombre de requêtes (et ne provoque pas de famine).

Pour minimiser les déplacements de bras lors d'accès répétitifs au fichier, on stocke les données du fichier sur les pistes appartenant à un même cylindre (ou sur des cylindres contigus pour les gros fichiers).

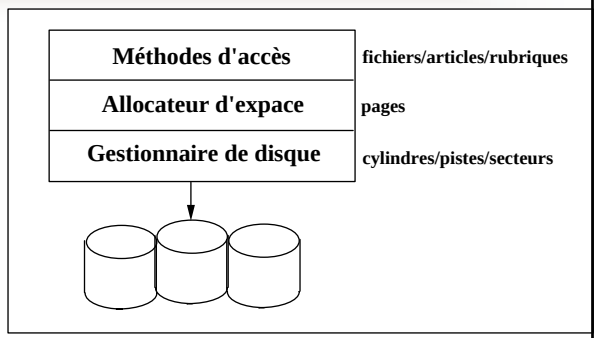
Exemple : disque à têtes fixes



Comparaison des stratégies

- Temps nécessaire (en nb de tours) pour exécuter les requêtes en attente selon la stratégie FIFO : 3 tours et 7 secteurs
- Séquences minimales : a) 4,2,5,1,3 b) 4,1,2,5,3 c) 5,2,4,1,3 d) 5,1,2,4,3 = 1 tour et 43 secteurs
- Séquence optimale : d
  - a et c diffèrent inutilement la requête 1
  - d traite plus vite la requête 5, permettant d'en traiter éventuellement une nouvelle qui arriverait ensuite
- Stratégie SLTF : 5,1,3,4,2 = 1 tour et 44 secteurs

3. Système de gestion de fichiers

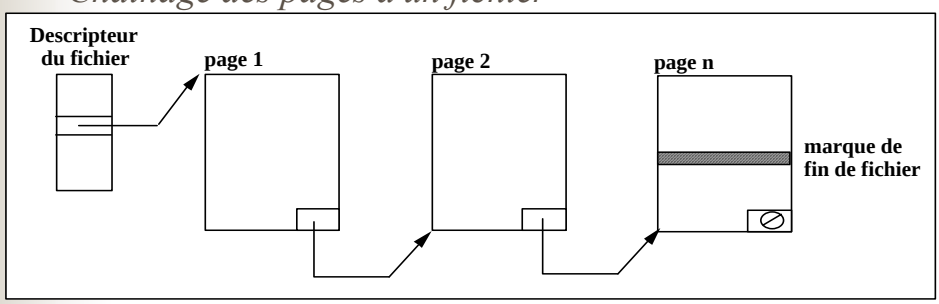


* Le SGF alloue à chaque fichier des pages mémoire, c'est-à-dire des granules de mémoire secondaire de taille fixe (par exemple 4K) pouvant être contigus ou non sur disque

* Le SGF traduit la clé d'article en adresse disque afin de commander l'E/S pour récupérer la page voulue d'un fichier

Structure de données gérée par l'allocateur d'espace

* *Chaînage des pages d'un fichier*



convient uniquement à la gestion de fichiers séquentiels

* Table d'allocation des pages (TAP)

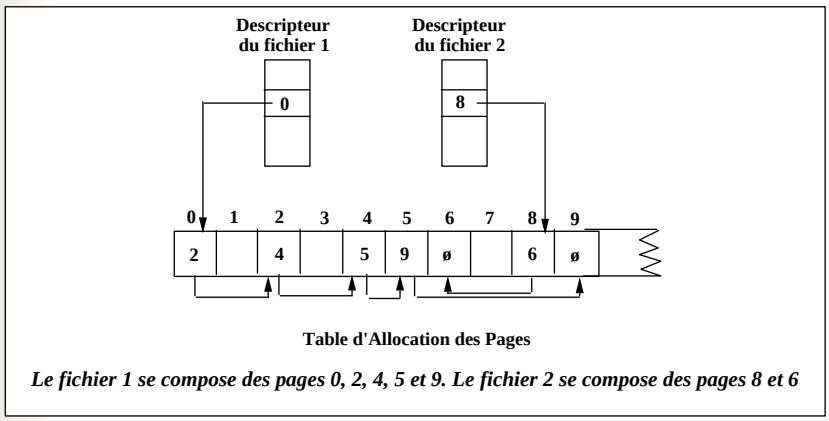
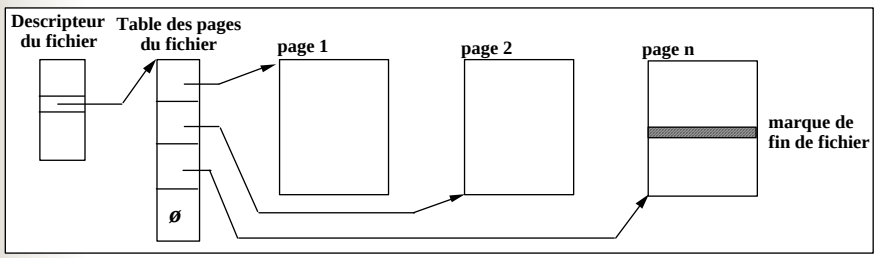


Table d'Allocation des Pages

Le fichier 1 se compose des pages 0, 2, 4, 5 et 9. Le fichier 2 se compose des pages 8 et 6

solution efficace si la TAP tient en mémoire, sinon comparable à la solution précédente

* Table de pages par fichier



c'est la meilleure technique pour la gestion des fichiers à accès directs (un accès direct à chaque page d'un fichier nécessite une seule E/S supplémentaire, pour lire la table des pages).

Un descripteur de fichier contient :

- le nom du fichier et l'identificateur du propriétaire,
- la liste des droits associés à ce fichier,
- la date de création et celle de dernière modification,
- la taille du fichier,
- la localisation du fichier (un pointeur sur la structure de données qui décrit l'ensemble des pages allouées à ce fichier),
- les paramètres spécifiant l'organisation du fichier.

4. Les concepts physiques d'Oracle

- Serveur de données Oracle =
  - BD + instance de serveur Oracle
- Instance =
  - ens. de processus d 'arrière plan et SGA
- SGA = System Global Area
    - DB_BLOCK_SIZE (taille en octets d 'un bloc svt 2 ou 4 Ko)
    - DB_BLOCK_BUFFERS (nb de tampons)
  - ens. de tampons mémoire dont
    - un tampon de données
    - un tampon de journal des transactions
  - partagés par les utilisateurs d'une BD

Les concepts d'Oracle (suite)

- Un fichier de contrôle / base
  - contient le nom de la base, la date de création, la localisation des espaces de données et des journaux de transactions
- Un fichier d'initialisation
  - contient les valeurs des paramètres
- Deux espaces physiques
  - images avant = rollback segments
  - images après = redo log files

Les concepts d'Oracle (suite)

- Une métabase par base
  - contient un espace SYSTEM
  - géré par un utilisateur SYS
  - 3 vues
    - user_ : objets de l 'utilisateur
    - dba_ : tous les objets
    - all_ : objets de l 'utilisateur + ceux auxquels il a accès
  - DICTIONARY : méta-métatable
- Structure des objets : commande DESCRIBE

Les concepts d'Oracle (suite)

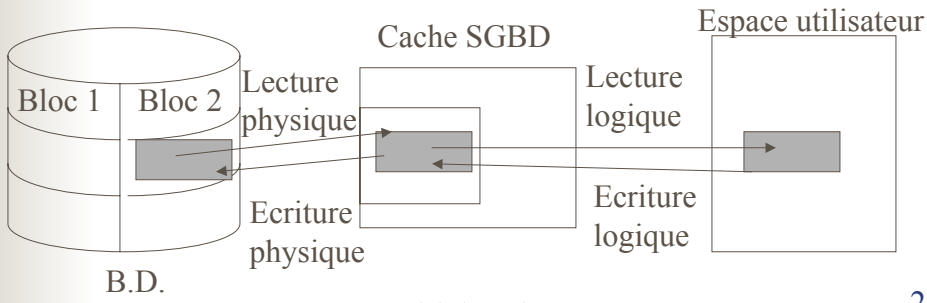
- Une base =
  - un ou plusieurs espaces logiques contenant :
    - méta-données
    - données
    - index
    - journaux
- Un espace logique = un ou plusieurs fichiers

Les concepts d'Oracle (suite)

- Un extent =
  - un espace constitué d'un nombre fixe ou variable de pages contiguës
  - l'unité d'allocation : quand on demande de l'espace, on obtient un extent
- Une page =
  - un en-tête, une suite de tuples (corps), une table de pointage des tuples
- Un tuple =
  - un en-tête = ROWID + nb col. NULL +…
  - la valeur effective du tuple avec pointeurs sur les objets longs
- Un espace logique = un ou plusieurs fichiers

Recherche d'un tuple

- Recherche de sa page
- si la page est dans le cache : lecture logique
- sinon, recherche dans le disque : lecture physique



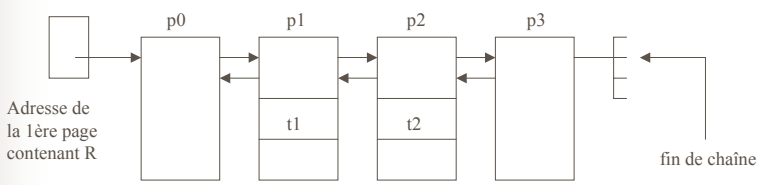
Ecriture de données

- Ecriture dans le tampon : écriture logique
- Ecriture dans le disque : écriture physique
- si la page est dans le cache : lecture logique
- sinon, recherche dans le disque : lecture physique

Gestion de la mémoire cache

- Si la page recherchée n'est pas dans le cache, elle est lue et rangée dans le cache
- Si toutes les pages du cache sont occupées, le système choisit la page à remplacer : avant de l'écraser, on la sauve sur les disques
  - 2 stratégies :
    - MRU Most Recently Used / LRU Less Recently Used
    - avec une liste de pages ordonnées selon leur dernière date d'utilisation
    - on peut mettre une marque de vidage pour écrire physiquement les pages au-delà, de façon asynchrone

Organisation en tas : HEAP



Organisation en tas : HEAP

- Recherche de tuples :
  - select * from T where condition ;
- On peut trouver l'adresse de la 1ère page qui contient les tuples de T
- la liste des pages contenant les tuples de T est parcourue par lectures successives (même si peu ou pas contiennent des tuples satisfaisant la condition)

Organisation en tas : HEAP

- Insertion de tuples :
  - dans la dernière page
  - sinon, dans une nouvelle page de l'extent
  - sinon, dans un nouvel extent
  - en général, on a un accès direct à la dernière page occupée
- Suppression de tuples :
  - on le localise et on décale les tuples suivants
  - si on libère une page, elle peut être utilisée pour d'autres tuples de la même table
  - si on libère un extent, on peut le réaffecter à une autre table de la BD

Organisation en tas : HEAP

- Mise à jour de tuples :
  - sans modification de taille : pas de changement de l'emplacement
  - avec réduction de taille : on décale les tuples suivants
  - avec augmentation de taille :
    - S'il y a assez de place dans la page : on décale
    - sinon, le tuple est supprimé de la page et inséré dans une autre page
  - parfois, suppression logique : le tuple n'est pas déplacé, on fait un chaînage

Organisation en tas : HEAP

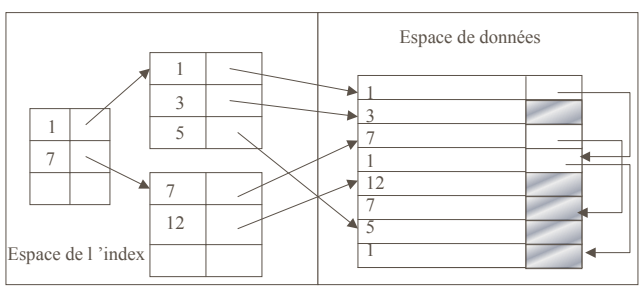
- C'est une organisation efficace pour :
  - les petites tables
  - ne nécessitant pas d'accès direct
  - subissant peu de modifications
- Nécessite des réorganisations périodiques

Organisation avec index

- La recherche des tuples peut être faite par un parcours séquentiel ou via des index
- Un index = structure de données qui associe à une valeur d 'un attribut (ou plusieurs), appelée clé de l'index, la ou les adresses des tuples contenant cette valeur

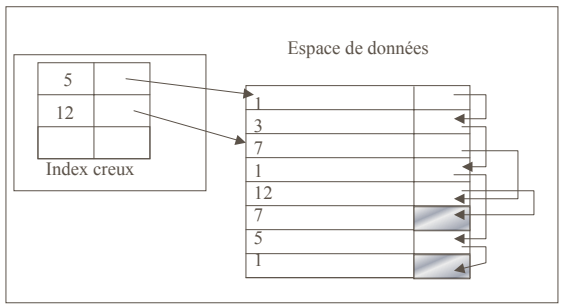
Organisation avec index

- Index dense (< > creux) : toutes les valeurs de clés sont dans la table



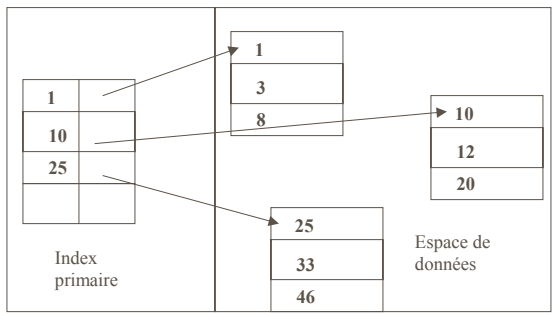
Organisation avec index

- Index creux : non dense



Organisation avec index

- Index primaire (< > secondaire) : défini sur une clé primaire, dont les valeurs sont ordonnées dans la table

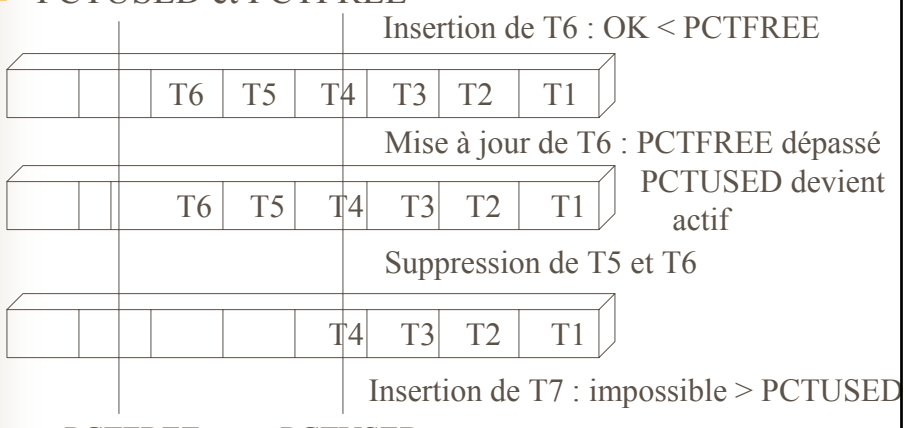


Contrôle de la densité des pages

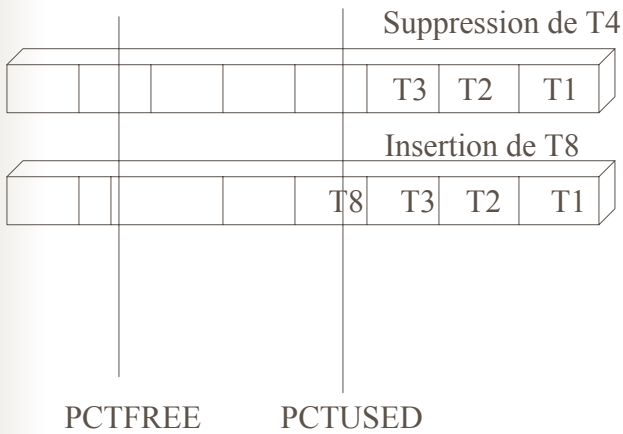
- Oracle gère des taux de remplissage
  - PCTUSED et PCTFREE
  - les insertions sont autorisées dans le bloc jusqu'à ce qu'il ne reste que PCTFREE d'espace libre
  - ensuite, seules les mises à jour sont autorisées dans ce bloc
  - les insertions sont de nouveaux permises lorsque le taux d 'occupation descend au-dessous de PCTUSED

Contrôle de la densité des pages

- On gère des taux de remplissage
  - PCTUSED et PCTFREE



Contrôle de la densité des pages (suite)



Les tablespaces

- Une BD = {tablespaces}
- Un tablespace = stocké dans un ou plusieurs fichiers de données (datafiles)
- Création, modification et suppression : commandes CREATE, ALTER et DROP
- Types :
  - SYSTEM (créé automatiquement)
  - affectés aux segments d 'annulation (rollback segments)
  - réservés au stockage des données et/ou des index
  - espaces temporaires de tri
  - temporaire (supprimé en fin de session) ou non

Un bloc Oracle

- Taille = multiple de la taille des blocs du système d'exploitation hôte
- descripteur de bloc (84 à 120 octets) + contenu

Un extent Oracle

- Groupe de blocs contigus
- option de tablespace :
  - UNIFORM : les extents sont de taille fixe (1Mo par défaut)
  - AUTOALLOCATE (par défaut) : le système choisit la taille des extents (par défaut et au minimum 64Ko)
  - exemple :
    - create tablespace T1 datafile « F:\Data\Fichier2T1.dat » size 5 M extent management local uniform size 128 K;
    - create tablespace T2 datafile « D:\Data\Fichier2T2.dat » size 2M extent management local uniform;

2 modes de gestion des tablespaces

- Mode local (management local) : une matrice d'occupation par datafile, gérée dans la tablespace
- Mode dictionnaire : on peut définir dans sa clause de stockage la taille de l'extent initial, celle de l'incrément, celle des suivants
  - exemple : create table T (x number(5),…) storage (initial 100K next 50K minextents 1 maxextents 30 pctincrease 5);
  - max 30 extents, le premier à une taille de 100K, le suivant de 50K, puis 5% de plus : si un bloc = 4K, 50K+5%=52.5K=13 blocs

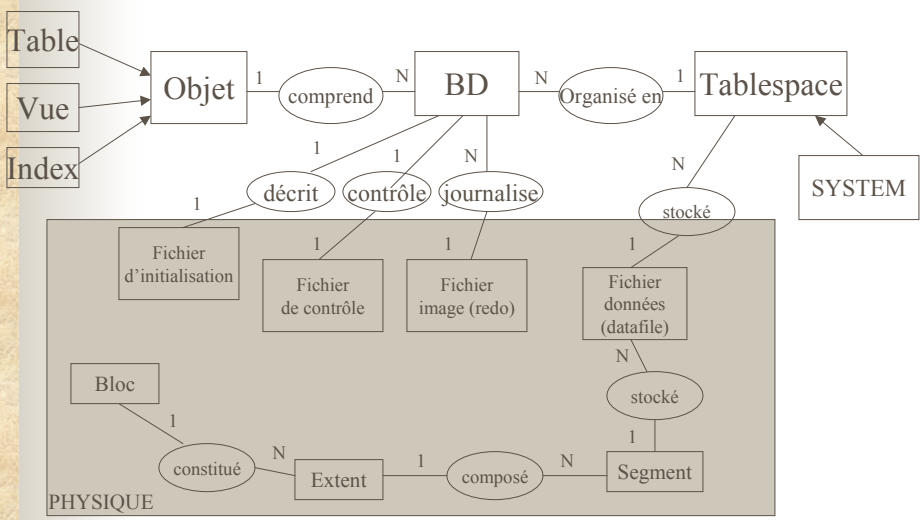
Segment Oracle

- Les extents d'un segment ne sont pas nécessairement contigus
- 4 types de segments :
  - segments de données
  - segments d'index
  - segments temporaires
  - segments d'annulation

Segments temporaires

- Utilisés pour les requêtes create index, select distinct, order by, group by, union, intersect, minus et jointure sans index
- par défaut, ils sont situés dans la tablespace SYSTEM, mais on peut les affecter à des tablespaces temporaires
- s'ils sont requis par une requête, ils sont libérés à la fin de l'exécution de la requête
- s'ils sont créés pour un index ou pour une table temporaire, ils sont supprimés à la fin de la session ou de la transaction qui a entraîné leur création

Synthèse des concepts principaux



Groupement de tables (cluster)

- Ranger dans un même bloc de données des tuples de différentes tables
- Rapprochement sur la base des valeurs de colonnes communes, appelées clés du cluster : les tuples ayant la même valeur de la clé sont rangés dans le même bloc

Exemple de cluster

- Regrouper un client et ses commandes
  - create cluster clico (ncli number(2)) size 400;
  - create table client (nocli number(2), nom varchar(30), …) cluster clico(nocli);
  - create table commande(ncom number(4),…,ncli number(2) not null) cluster clico(ncli);
  - create index ic on cluster clico;

Exemple de cluster

ncli
12

Nomcli	Adr
Jules	Aix

Ncom	Montant	Datec	…
1234	12000	12/12/03	
1279	18005	18/12/03	
…			

ncli
18

Nomcli	Adr
Jean	Paris

Ncom	Montant	Datec	…
2444	2000	12/12/03	
8279	8005	18/12/03	
…			

Principe

- Clé de cluster : jusqu'à 16 colonnes, sauf type LONG ou RAW
- SIZE : taille moyenne de l'espace requis pour accueillir la liste des tuples associée à une valeur de la clé du cluster
- Un index sur la clé du cluster est obligatoire
- Il doit être créé après le CREATE CLUSTER et avant toute insertion de données dans les tables

5. Démarche de Conception Physique

La conception de structures physiques implique le choix :

- des structures pour les types d'enregistrements ou de relations,
- des types de liens devant exister entre les enregistrements (pour les bases de données navigationnelles).

Les étapes

A- Choix d'une structure initiale

B - Vérification de la structure

C - Validation

D - Restructuration

E - Choix des organisations primaires

F - Choix des organisations secondaires

A. Choix d'une structure initiale pour les enregistrements

Ce choix d'une structure correcte est important. En effet,

- des enregistrements trop petits vont entraîner un grand nombre de jointures dans les systèmes relationnels, d'opérations navigationnelles dans les autres SGBD,
- des enregistrements trop grands nécessitent des transferts de données excessifs et une utilisation non optimale des zones de tampon.

B. Vérification des structures

La vérification de la structure d'une base de données consiste à s'assurer que :

- chaque type d'enregistrement a un nom unique,
- même chose pour les champs d'un enregistrement,
- même chose pour les liens dans les SGBD navigationnels,
- chaque champ de chaque enregistrement est associé à un type de données.

C. Validation de la structure

Il s'agit de prouver qu'un choix de conception satisfait aux performances relatives aux ressources et aux temps de réponse.

La validation comprend :

- le calcul des besoins de stockage,
- le calcul des performances de traitement.

Calcul des besoins de stockage

1. *Les types de données*

   A chaque type de donnée affecté à chaque champ, il faut associer une taille moyenne et une taille maximum.

   Ces estimations seront utilisées pour calculer le besoin de stockage.

2. *Les volumes des enregistrements*

   Là encore, on estime le nombre moyen d'enregistrements de chaque type de la future base de données ainsi que le nombre maximum.

3. *Les constantes de la base de données*

   Un certain nombre de grandeurs sont caractéristiques du SGBD utilisé, par exemple la partie fixe (overhead) nécessaire au stockage d'un type d'enregistrement ainsi que la partie fixe (overhead) nécessaire au stockage d'un enregistrement.

   Les champs ont des longueurs variables. Il faut donc estimer l'espace occupé par les séparateurs.

   Enfin, pour chaque SGBD, on estime la taille occupée par un type de données. Par exemple, les champs numériques nécessitent en général 1 octet pour 2 caractères numériques plus 1 octet pour l'exposant et le signe.

4. *Besoin moyen en stockage pour les enregistrements*

Pour chaque type d'enregistrement, la taille moyenne des données peut être estimée par la formule suivante :

taille moyenne de l'enregistrement * nombre moyen d'enregistrements

+ overhead du type d'enregistrement

La taille moyenne de l'enregistrement peut, quant à elle, être estimée par la formule suivante :

longueur moyenne des champs
+ longueur des séparateurs * nombre de séparateurs
+ overhead de l'enregistrement

Il faut ajouter l'espace occupé par les index. On estime grossièrement qu'il est égal au tiers de l'espace occupé par les données.

5. *Besoin maximal en stockage pour les enregistrements*

En utilisant les données maximales, on peut de la même façon calculer l'espace maximal occupé.

6. *Besoin total en stockage*

Ce travail est répété pour tous les types d'enregistrements et conduit à l'estimation du besoin total en stockage pour la future base.

7. *Taux de croissance de la base*

Pour cela, il faut connaître les principales mises à jour par unité de temps.

Pour chaque type d'enregistrement, on estime le taux de croissance, puis le taux de croissance global de la base.

Calcul des performances de traitement

On établit la liste des transactions avec :

- leur fréquence,
- les opérations de l'algèbre relationnelle nécessaires selon la structure choisie,
- la priorité des transactions, qui peut être estimée par :

$$\sum_i \text{fréquence transaction } T_i * \text{nombre d'opérations relationnelles de } T_i$$

D. Restructuration

1. Les opérateurs de restructuration

1.1. La composition

On peut réaliser une composition dans l'un des trois cas suivants :

a) Cas où les structures ont des identifiants communs

C'est une composition qui s'apparente à une union ou à une jointure externe selon le contenu des tables initiales.

Exemple :

R1 = Fournisseurs
(Code fournisseur, Nom, Adresse)

Code fourniss.	*Nom Fourniss.*	*Adresse fourniss.*
1234	DUPONT	Paris
1235	DUVENT	Marseille
1237	DUPUIS	Lyon
1238	DULAIT	Grenoble

R2 = Produits
(Code fournisseur, Désignation)

Code fournisseur	*Désignation produit*
1234	Pneus
1236	Roues
1238	Guidons

Composition R3 = Fourprod (Code fournisseur, Nom, Adresse, Désignation)

Code fournisseur	*Nom fournisseur*	*Adresse fournisseur*	*Désignation produit*
1234	DUPONT	Paris	Pneus
1235	DUVENT	Marseille	N/A
1236	N/A	N/A	Roues
1237	DUPUIS	Lyon	N/A
1238	DULAIT	Grenoble	Guidons

b) Cas où l'identifiant unique d'une structure est un sous-ensemble de l'autre structure et où la composition se fait sur ce sous-ensemble commun

Exemple :

SOINS(code médecin,code patient)

MEDECINS(code médecin, nom médecin, spécialité)

Code médecin	*Code patient*
1234	4327
1235	4326
1237	4235
1238	4234

Code médecin	*Nom médecin*	*Spécialité médecin*
1234	DUPONT	cardiologie
1235	DULAIT	pédiatrie
1236	DUVIN	urologie
1237	DELEAU	cancérologie
1238	DELAIR	hématologie

Composition SOINS-MED : (code médecin, code patient, nom médecin, spécialité)

Code médecin	*Code patient*	*Nom médecin*	*Spécialité médecin*
1234	4327	DUPONT	cardiologie
1235	4326	DULAIT	pédiatrie
1236	N/A	DUVIN	urologie
1237	4235	DELEAU	cancérologie
1238	1234	DELAIR	hématologie

c) Cas où la composition se fait sur une clé étrangère ne faisant pas partie de l'identifiant unique :

Exemple :

Règlement(code-règlement, date-règlement, code-facture)

Facture(code-facture, date-facture, montant-facture)

se composent en :

Règl-factures (code-règlement, date-règlement, code-facture, date-facture, montant-facture)

La composition permet de restructurer des enregistrements, notamment des enregistrements de petite taille.

Si les structures qui subissent une composition sont normalisées, le résultat de cette composition n'est pas nécessairement normalisé.

1.2. La décomposition verticale

La décomposition verticale permet de décomposer une grande structure en structures plus petites.

Ces dernières possèdent alors un sous-ensemble de champs de la structure d'origine.

Il s'agit de projections au sens relationnel.Chaque structure résultante doit posséder l'identifiant de la structure d'origine.

Considérons l'exemple suivant :

Clients (code-client, nom-client, adresse-client, nom-produit)

On peut décomposer Clients verticalement pour obtenir :

Clients-1 (code-client, nom-produit)

Clients-2 (code-client, nom-client, adresse-client)

1.3. La décomposition horizontale

La décomposition horizontale permet d'obtenir des sous-ensembles d'enregistrements d'une même structure.

Il s'agit d'un ensemble de restrictions au sens relationnel.

Il faut que tout enregistrement de la structure d'origine appartienne à au moins l'une des structures résultant de la décomposition. Au sens relationnel, l'union de l'ensemble des tables doit être la structure initiale.

Exemple : Soit la structure

Clients (code-client, nom-client, adresse-client, chiffre-d-affaires)

On veut décomposer la structure selon que le montant du chiffre-d-affaires est inférieur ou supérieur à 50000 €

On obtient :

Clients-riches (code-client, nom-client, adresse-client, chiffre-d-affaires)

Clients-moins-riches (code-client, nom-client, adresse-client, chiffre-d-affaires)

1.4. L'aplatissement

Cet opérateur n'a pas d'équivalent en algèbre relationnelle. Il permet de transférer des données d'un niveau de représentation à un autre.

Il permet par exemple de transférer des données appartenant à des valeurs de champs à des noms de champs. Cela ne s'applique qu'à des champs discrets avec un nombre limité de valeurs.

Exemple : Soit la structure :

Cours (code-cours, nom-cours, jour-cours, heure-cours)

Cette structure peut être aplatie suivant le jour pour donner :

Cours (code-cours, nom-cours, heure-cours-lundi,
heure-cours-mardi, heure-cours-mercredi,
heure-cours-jeudi,heure-cours-vendredi,
heure-cours-samedi)

2. *La matrice des transactions*

Elle donne, pour chaque transaction, les champs utilisés et leur type d'utilisation :

	TRANS. N°1 (FREQUENCE)	TRANS. N°2 (FREQUENCE)	.....	TRANS. N°n (FREQUENCE)
CHAMP 1	MODIFIE	R		X
CHAMP 2	M	R		R
CHAMP n	X			X
. . .				

M indique que le champ est modifié par la transaction,

R indique que le champ est utilisé en sortie par la transaction,

X indique que le champ est utilisé pour un calcul par la transaction.

3. *Analyse de la matrice des transactions*

- *analyse par groupe de champs :*

pour déterminer si des compositions ou décompositions permettent de diminuer le nombre d'opérations des transactions fréquentes.

- *analyse des formes normales temporelles :*

tout ensemble de champs dont les valeurs existent ensemble et qui cessent d'avoir un intérêt simultanément devraient être restructurées. Dans la matrice de transactions, ce sont les champs indiqués par M et utilisés lors de mises à jours.

- *analyse des clés :*

pour diminuer l'espace occupé par les fichiers tout en améliorant la fiabilité d'accès aux données, on peut créer des clés uniques au lieu de clés composées ou remplacer des clés longues par des codes plus courts et plus fiables.

- *analyse des dérivations :*

il s'agit d'estimer s'il est préférable de stocker le résultat d'un calcul (ce qui introduit une redondance) ou de le calculer chaque fois que nécessaire.

4. Recommandations

On effectue la synthèse des différentes analyses pour suggérer une restructuration des enregistrements de la table.

5. Evaluation des bénéfices des restructurations

On établit la matrice suivante :

	TRANS. # 1 (FREQUENCE)	TRANS. # 2 (FREQUENCE)	.....	TRANS. # n (FREQUENCE)
Recommandation # 1	+	-		-
Recommandation # 2	-	+		-
...				

pour évaluer l'impact de chaque recommandation sur l'ensemble des transactions.

On en déduit le sous-ensemble des recommandations qui améliorent globalement les performances des transactions.

C'est ce sous-ensemble qui va donner lieu à la restructuration. Ce processus peut éventuellement être réitéré.

E - Choix des organisations primaires

A partir de cette matrice, on peut calculer pour chaque type d'enregistrement :

- la proportion d'accès par la clé primaire unique
- la proportion de tris sur la clé primaire.

La matrice ci-dessous permet de guider le choix d'une organisation primaire à partir de ces proportions :

Cette table doit être complétée et/ou modifiée selon le SGBD et les organisations offertes.

	ACCES UNIQUE	***ACCES SEQUENTIEL***
HEAP	< 5%	< 5%
HASH	> 50%	> 5%
SEQUENTIEL	< 5%	< 50%
SEQUENTIEL INDEXE	> 5%	> 50%

F - Choix des organisations secondaires

Le choix d'une organisation secondaire dépend aussi du type de SGBD utilisé :

- index secondaires dans les systèmes relationnels,
- liens bidirectionnels et «sets system» pour les systèmes réseaux.

On établit une matrice d'accès secondaire, avec :

- en colonne, les transactions et leurs fréquences,
- en ligne, les champs non-clés de chaque type d'enregistrement,
- à l'intersection, on marque par X les transactions qui requièrent l'accès direct à l'aide d'un champ non clé.

Ils sont candidats pour devenir des index secondaires.

6. Exemple d'application

1. Choix d'une structure initiale

Flights (flight_number/char(6); aircraft/char(30), distance/integer, airline_code/char(3))

Airlines (airline code/char(3);airline_name/char(30))

Times (flight_number/char(6), day/char(3);dep_time/decimal(4,2), arr_time/decimal(4,2))

Stops (flight_number/char(6), airport code/char(3); stop_number/integer)

Airports (airport_code/char(3); airport_name/char(30),country_code/char(4))

Countries (country_code/char(4);country_name/char(30),continent_name/char(30), restrictiveness/char(30))

Restrictions (country_code/char(4), visa_type/char(30);conditions/text)

Fares (flight_number/char(6), type_code/char(4), conc_class/char(30);
single/decimal(6,2), return/decimal(6,2))

Types (type_code/char(4);seat_class_code/char(1), saver_code/char(3), season_code/char(1))

Seat_Classes (seat_class_code/char(1); seat_class_name/char(30))

Season (season_code/char(1); season_name/char(30)), start_date/dates, end_date/dates)

Savers (saver_code/char(3); saver_name/char(30), saver_conditions/text)

2. Vérification

La structure proposée semble être cohérente suivant les règles de nom

3. Validation

Il faut calculer les besoins de stockage et les performances

3.1. Les besoins de stockage

3.1.1. Les types de données

Domain name	*Type*	*Mean size*	*Maximum size*
codes	char	4	6
distances	integer	4	5
days	char	3	3
times	decimal	4	4
stop_numbers	integer	1	1
names	char	15	30
text	char	50	240
money	decimal	5	6
date	dates	6	6

3.1.2. Les volumes des enregistrements

Record type	*Mean volume*	*Maximum volume*
Flights	20.000	25.000
Airlines	100	150
Times	22.500	30.000
Stops	50.000	75.000
Airports	500	750
Countries	75	100
Restrictions	150	250
Fares	200.000	300.000
Types	5	20
Seat_Classes	3	3
Seasons	2	2
Savers	3	5

3.1.3. Les constantes de la base de données

- Overhead du type d'enregistrement : 6 Ko (6144).
- Overhead de l'enregistrement : 5 octets.
- Les champs ont des longueurs variables. Les séparateurs de champ requièrent 1 octet chacun.
- Les champs numériques nécessitent 1 octet pour 2 caractères numériques plus 1 octet pour l'exposant et le signe.

3.1.4. Besoin moyen en stockage pour les types d'enregistrements (ici Flights)

Taille moyenne des données du type d'enregistrement = overhead du type d'enregistrement + taille moyenne de l'enregistrement * nombre moyen d'enregistrements

Taille moyenne de l'enregistrement = overhead de l'enregistrement + longueur moyenne du code + longueur du séparateur + longueur moyenne du nom + longueur du séparateur + longueur moyenne de la distance + longueur du séparateur + longueur moyenne du code

= 5 + 4 + 1 + 15 + 1 + 3 + 1 + 4 = 34 octets

taille moyenne des données	=	6144 + (34 * 20 000) = 670 Ko
taille de l'index	=	taille du type d'enregistrement * (1/3) = 224 Ko
taille moyenne du type d'enregistrement	=	(taille moyenne des données du type d'enregistrement + taille index) * facteur d'utilisation
	=	(670 + 224) * 4/3 = 1192 Ko

3.1.5. Besoin maximal en stockage pour les types d'enregistrements (ici Flights).

taille maximale du type d'enregistrement	=	2355 Ko

3.1.6. Besoin total en stockage

Record type	*Mean size*	*Maximum size*
Flights	1.192	2.355
Airlines	16	23
Times	832	1210
Stops	1.487	2.746
Airports	38	75
Countries	19	30
Restrictions	31	135
Fares	14.595	30.740
Types	12	12
Seat_Classes	12	12
Seasons	12	12
Savers	12	14
Total	*18.258*	*37.364*

3.1.7. Taux de croissance de la base

Pour cela il faut connaître les mises à jour. Parmi les principales, notons :

x1 :ajouter les informations relatives à un nouveau vol (500 par an soit 1,5 par jour)

x2 : supprimer les informations relatives à un vol (350 par an = 1 par jour)

x3 :modifier les données relatives au temps de vol (1000 par an = 3 par jour)

Le taux de croissance de 150 vols par an vous amène à considérer un taux de croissance de la base égal à :

150 enregistrements Flight = 34 x 150 = 5 100 octets

(22 500/20 000) * 150 enregistrements Times = 168.75 * 21= 3 544 octets

(50 000/20 000) * 150 enregistrements Stops = 375 * 17 = 6 375 octets

(20 000/20 000) * 150 enregistrements Fares = 1 500 * 42 = 63 Ko

= 78 019 octets

La croissance se situe entre 0.5 et 1% de sa taille totale.

3.2. Calcul des performances

Pour cela, il faut connaître les transactions principales. En plus des transactions de mise à jour considérées plus haut (x1, x2,x3), nous avons :

x4 : Lister les tarifs pour tous types de tickets disponibles, quelle que soit la compagnie aérienne, pour voyager de l'aéroport A à l'aéroport B (500 par jour)

x5 : Heures d'arrivée et de départ du vol V (500 par jour)

x6 : Pour un vol V, lister les noms des aéroports des Stops intermédiaires (300 par jour)

x7 : Les restrictions d'entrée applicables au pays P (100 par jour)

3.2.1.Expression des transactions sous forme d'algèbre relationnelle

X1 Flights ← union (Flights, NewFlight)
Times ← union (Times, NewFlightTimes)
Stops ← union (Stops, NewFlightStops)
Fares ← union (Fares, NewFlightFares)
X2 Flights ← difference (Flights, OldFlight)
Times ← difference (Times, OldFlightTimes)
Stops ← difference (Stops, OldFlightStops)
Fares ← difference (Fares, OldFlightFares)
X3 Times ← union (difference (Times, OldFlightTimes), NewFlightTimes)
X4 T1 ← join (Stops, Airports; airport_code)
T2 ← restrict (T1; stop_number = 0 and airport_name = A)
T3 ← restrict (T1; stop_number > 0 and airport_name = B)
T4 ← intersect (T2, T3)
T5 ← join (T4, Fares; flight_number)
T6 ← project (T5; flight_number, type_code, conc_class, single, return)

3.2.1.Expression des transactions sous forme d'algèbre relationnelle (suite)

X5 : T1 ← restrict (Times, flight_number = F)
X6 : T1 ← restrict (Stops; flight_number = F and stop_number >0)
T2 ← join (T1, Airports; airport_code)
T3 ← project (T2; airport_name, stop_number)
X7 : T1 ← restrict (Countries; country_name = C)
T2 ← join (T1, Restrictions; country_code)
T3 ← project (T2; visa_type, conditions)

3.2.2. La charge des transactions

	X1 (1.5)	X2 (1)	X3 (3)	X4 (500)	X5 (500)	X6 (300)	X7 (100)
product	0	0	0	0	0	0	0
union	4	0	1	0	0	0	0
difference	0	4	1	0	0	0	0
intersection	0	0	0	1	0	0	0
restriction	0	0	0	2	1	1	1
project	0	0	0	1	0	1	1
join	0	0	0	2	0	1	1
divide	0	0	0	0	0	0	0
Total	4	4	2	6	1	3	3

La priorité des transactions selon leur fréquence
(x4, x5), x6, x7, x3, x1, x2
Si l'on tient des opérateurs, la liste devient x4, x6, x5, x7

4. Les considérations de restructuration

4.1. La matrice des transactions

	X1 (1.5)	X2 (1)	X3 (3)	X4 (500)	X5 (500)	X6 (300)	X7 (100)
Flight_number	M	M	X	R	X	X	
aircraft	M	M					
distance	M	M					
airline_code	M	M					
airline_name							
day	M	M	M		R		
dep_time	M	M	M		R		
arr_time	M	M	M		R		
airport_code	M	M		X		X	
stop_number	M	M		X		X	
airport_name				X		R	
country_code							X
country_name							X
continent_name							
visa_type							R
conditions							R
type_code	M	M		R			
conc_class	M	M		R			
single	M	M		R			
return	M	M		R			
seat_class_code							
saver_code							
season _code							
seat_class_name							
start_date							
end_date							
saver_name							
saver_conditions							

4.2. Les techniques de restructuration

4.2.1. Analyse par groupe de champs

- Pas de décomposition suggérée
- Les compositions proposées sont :
  - Countries et Restrictions pour introduire country_name dans Restrictions, comme requis par la transaction x7
  - Stops et Airports pour introduire airport_name dans Stops comme requis par les transactions x4 et x6
  - Stops et Fares suggérées par les transactions x1, x2, x4

4.2.2. Analyse des formes normales temporelles

- Deux compositions sont suggérées par les transactions x1, x2, x3
  - Fligths, Stops et Fares
  - Figths, Times, Stops et Fares
- Ces mêmes transactions suggèrent 2 aplatissements
  - Stops sur stop_number
  - Times sur day

4.2.3. Analyse des clés

Il y a suffisamment de clés d'accès

4.2.4. Analyse des dérivations

Il n'y a aucune valeur dérivée dans cet exemple

4.2.5. Les recommandations

Au total, nous recommandons :

R1 : Composition de Countries et Restrictions (Trans. x7)
R2 : Composition de Stops et Airports (Trans. x4 et x6)
R3 : Composition de Stops et Fares (Trans. x1, x2, x4)
R4 : Composition de Stops, Fares, Flights (Trans. x1, x2)
R5 : Aplatissement de Stops (Trans. x1)
R6 : Aplatissement de Times (Trans. x1, x3)

4.2.6. Bénéfices des restructurations

	X1	X2	X3	X4	X5	X6	X7
R1							+
R2	-	-		+		+	
R3	+		-				
R4	+	+	-	-	-	-	-
R5	+		+				
R6	+	+	+		+		

En effet, la composition de Stops, Flights et Fares, va désavantager x3, x4, x5, x6 en exigeant de traiter des enregistrements de plus grande taille qu'elles ne nécessitent.

4.2.7. Nouvelle liste des recommandations

R1, R2, R3, R5,R6 globalement positives

R4 positive-négative -> à supprimer

4.2.8. La nouvelle structure

Flights (flight_number/codes; aircraft/names, distance/distances, airline_code/codes)

Airlines (airline_code/codes; airline_name/names)

*Times (flight_number/char(6); mon_dep/decimal(4,2),mon_arr/decimal(4,2), tue_dep/decimal(4,2), …,sun_dep/decimal(4,2), sun_arr/decimal(4,2))

*Int_Stops (flight_number/char(4), airport_name/char(30); stop_number/integer)

*End_Stops (flight_number/char(4); origin/char(30), destin/char(30))

Airports (airport_code/codes; airport_name/names country_code/codes)

*Countries (country_code/codes; country_name/names, continent_name/names)

*Restrictions (country_name/char(30), visa_type/char(30); conditions/text)

Fares (flight_number/char(6), type_code/char(4), conc_class/char(30);single/decimal(6,2), return/decimal(6,2))

Types (type_code/char(4);seat_class_code/char(1),saver_code/char(3),season_code/char(1))

Seat_Classes (seat_class_code/char(1); seat_class_name/char(30))

Seasons (season_code/char(1); season_name/char(30), start_date/dates, end_date/dates)

Savers (saver_code/char(3); saver_name/char(30))

4.2.8.1. Vérification OK

4.2.8.2. Validation

4.2.8.2.1. Le stockage

En évaluant la nouvelle structure, on obtient

Record type	***Mean size***	***Maximum size***
Flights	1.192	2.355
Airlines	16	23
Times	467	1364
Int_Stops	498	2624
End_Stops	1.435	2.546
Airports	38	75
Countries	15	20
Restrictions	34	144
Fares	14.595	30.740
Types	12	12
Seat_Classes	12	12
Seasons	12	12
Savers	12	14
Total	*18.338*	*39.941*

4.2.8.2.2. Les performances

	X1 (1.5)	X2 (1)	X3 (3)	X4 (500)	X5 (500)	X6 (300)	X7 (100)
product	0	0	0	0	0	0	0
union	5	0	1	0	0	0	0
difference	0	5	1	0	0	0	0
intersection	0	0	0	0	0	0	0
restriction	0	0	0	1	1	1	1
project	0	0	0	1	0	0	0
join	0	0	0	1	0	0	0
divide	0	0	0	0	0	0	0
Total	5	5	2	3	1	1	1

On constate une nette amélioration

5. Choix de l'organisation interne

5.1. Hypothèses sur le SGBD

- offre l'organisation HEAP dynamique
- offre l'organisation séquentielle-indexée pour les organisation primaires
- offre des index non-uniques pour les organisations secondaires

5.2. Choix des organisations primaires

5.2.1. Cas des sites répartis (essentiellement des interrogations)

	X1(X4) (2335)	X2(X5) (2335)	X3(X6) (1400)	X4(X7) (465)
Flights				
Airlines		U		
Times			U	
Int_Stops				
End_Stops	U			
Airports				
Countries				
Restrictions				U
Fares	U			
Types				
Seat_Classes				
Seasons				
Savers				

	Unique primary key	***Sorted primary key***	***Non-primary key***
Flights	0%	0%	100%
Airlines	0%	0%	100%
Times	36%	0%	64%
Int_Stops	21%	0%	79%
End_Stops	36%	0%	64%
Airports	0%	0%	100%
Countries	0%	0%	100%
Restrictions	7%	0%	93%
Fares	36%	0%	64%
Types	0%	0%	100%
Seat_Classes	0%	0%	100%
Seasons	0%	0%	100%
Savers	0%	0%	100%

- *Les recommandations*
  - HEAP

    Flights, Airlines, Airports, Countries, Types, Seat-Classes, Seasons, Savers

    Raison : Besoins d'accès par la clé primaire est bas
  - SEQ.IND

    Times, Int-Stops, End-Stops, Restrictions, Fares

    Raison : Nécessite d'avoir un accès direct par la clé primaire, et c'est la seule organisation primaire autorisée

5.2.2. Cas du site central (essentiellement de UPDATE)

	X1 (500)	X2 (350)	X3 (1000)
Flights		U	
Airlines			
Times		U	U
Int_Stops		U	
End_Stops		U	
Airports			
Countries			
Restrictions			
Fares		U	
Types			
Seat_Classes			
Seasons			
Savers			

	Unique primary key	***Sorted primary key***	***Non-primary key***
Flights	19%	0%	81%
Airlines	0%	0%	100%
Times	73%	0%	27%
Int_Stops	19%	0%	81%
End_Stops	19%	0%	81%
Airports	0%	0%	100%
Countries	0%	0%	100%
Restrictions	0%	0%	100%
Fares	19%	0%	81%
Types	0%	0%	100%
Seat_Classes	0%	0%	100%
Seasons	0%	0%	100%
Savers	0%	0%	100%

Les recommandations

HEAP

Airlines, Airports, Countries, Restrictions, Types, Seat-Classes, Seasons, Savers

SEQ.IND

Flights, Times, Int-Stops, End-Stops, Fares

5.3. Choix des organisations secondaires

5.3.1. Les sites répartis

		X1(X4) (2335)	X2(X5) (2335)	X3(X6) (1400)	X4(X7) (465)
End_Stops	origin	X			
	destin	X			

36% accès direct à End-Stops à partir des valeurs "origine" et "Destin" =>
choisir un index composé (origine, destination) plutôt que deux index séparés

5.3.2. Le site central

Pas d'accès secondaire => pas d'organisation secondaire

Conclusion

- Pour améliorer les performances d'une base de données, l'administrateur dispose de différents leviers :
  - le schéma logique peut être restructuré
  - les structures physiques (tablespace, index, cluster, …) peuvent être ajoutées, modifiées, supprimées
  - les requêtes et programmes peuvent être optimisées (cf évaluation de requêtes)