

Stratégies de gestion de données relationnelles

*Ce support de cours est basé sur celui de M. Jacques LE MAITRE,
Ancien Professeur à l'USTV, que je remercie pour son aide précieuse.*

Sommaire

PRÉSENTATION

1	Introduction générale	3
1.1	Petit retour à la base (de données !) ...	4
1.2	Paradigmes de gestion de données.....	5
1.3	Architectures de BD/SGBD	10
1.4	Pourquoi un SGBD ?	20
1.5	OS et SGBD, même combat ?	22

LIVRE 1 : ASSURER LE CONTRÔLE DES DONNÉES

2	Introduction au contrôle des données	24
2.1	Le souci du contrôle des données	25
2.2	Vers le concept de « transaction » et les propriétés ACID.....	26
2.3	Assurer la pérennité des données	26
3	Gestionnaire transactionnel	27
3.1	Notion de transaction	29
3.2	Propagation des modifications.....	32
3.3	Problèmes dus à la concurrence.....	34
3.4	Contrôle de concurrence.....	40
3.5	Paramètres du gestionnaire transactionnel	74
4	Gestionnaire de reprise après panne	75
4.1	Types de pannes.....	76
4.2	Reprise « intuitive »	77
4.3	Annulation de transactions en cascade	80
4.4	Annulations en cascade : traitement « intuitif » et « évitement »	81
4.5	Surveillance des transactions : journalisation	82
4.6	Reprise après panne : utilisation du journal de reprise	85
4.7	Paramètres de la journalisation	88

LIVRE 2 : OFFRIR DES PERFORMANCES OPTIMALES

5	Introduction à l'optimisation.....	90
5.1	Stockage physique et logique (OS) des données	92
5.2	Traitement d'une requête	108
6	Organisation physique	109
6.1	Organisation interne générale des pages de stockage des n-uplets.....	111
6.2	Adressage d'un n-uplet	114
6.3	Stockage des données des n-uplets.....	116
6.4	Stratégies d'adressage des n-uplets	121
6.5	Stratégies de stockage des valeurs d'attributs	133
6.6	Stratégies de gestion des réertoires des déplacements	135
6.7	Impact des différentes stratégies au niveau des performances	138
6.8	Paramètres de l'organisation physique	141
7	Gestionnaire de mémoire cache	146
7.1	Principe de la mémoire cache	148
7.2	Structure de la mémoire cache	149
7.3	Utilisation de la mémoire cache	149
7.4	<i>Quid</i> de l'impact sur les performances ?	158
8	Indexation des données	159
8.1	Notions de base sur les index.....	162
8.2	Structure d'un index.....	171
9	Optimisation de requêtes.....	248
9.1	Optimisation pré-évaluation : choix de la meilleure stratégie.....	252
9.2	Prise en compte de l'optimisation pendant l'évaluation : les pipelines	321
9.3	Paramètres liés à l'optimisation de requêtes	328
SYNTHESE		
10	Récapitulatif de l'ensemble des paramètres.....	330
10.1	Paramètres liés au contrôle des données (livre 1).....	331
10.2	Paramètres liés à l'optimisation (livre 2).....	331

1 Introduction générale

SOMMAIRE DÉTAILLÉ DU CHAPITRE 1

1.1	Petit retour à la base (de données !).....	4
1.2	Paradigmes de gestion de données	5
1.2.1	Les principaux paradigmes de gestion de données	5
1.2.2	Le paradigme de ce support : le relationnel	7
1.2.2.1	Relations, attributs et n-uplets	7
1.2.2.2	Constituants	7
1.2.2.3	Clés (primaires et étrangères)	8
1.3	Architectures de BD/SGBD.....	10
1.3.1	Architecture(s) conceptuelle(s)	10
1.3.1.1	Schéma interne (ou « schéma physique »).....	11
1.3.1.2	Schéma conceptuel	11
1.3.1.2.1	Dans le cadre du paradigme relationnel	12
1.3.1.2.2	Dans le cadre du paradigme objet.....	15
1.3.1.2.3	Dans le cadre du paradigme documentaire	16
1.3.1.3	Schéma externe	17
1.3.1.4	Pourquoi ces 3 niveaux ? L'indépendance données/traitements	18
1.3.2	Architecture logique	18
1.4	Pourquoi un SGBD ?.....	20
1.5	OS et SGBD, même combat ?.....	22

FIGURES DU CHAPITRE 1

Figure 1.	SGBD et BD	4
Figure 2.	Concepts basiques du paradigme relationnel	7
Figure 3.	Architecture ANSI/SPARC à 3 niveaux	10
Figure 4.	Schéma conceptuel relationnel	13
Figure 5.	Instance de schéma conceptuel relationnel	13
Figure 6.	Exemple de schéma conceptuel relationnel	14
Figure 7.	Exemple d'instance de schéma conceptuel relationnel	14
Figure 8.	Exemple de schéma conceptuel orienté objet	15
Figure 9.	Exemple d'instance de schéma conceptuel orienté objet	15
Figure 10.	Exemple d'instance de schéma conceptuel orienté objet (autre vision)	16
Figure 11.	Exemple de schéma conceptuel documentaire	16
Figure 12.	Exemple d'instance de schéma conceptuel documentaire	17
Figure 13.	Architecture logique type d'un SGBD	19
Figure 17.	Principaux mécanismes d'un SGBD abordés dans ce cours	21

TABLEAUX DU CHAPITRE 1

Tableau 1.	Apports des principaux mécanismes d'un SGBD	22
------------	---	----

ÉQUATIONS DU CHAPITRE 1

Aucune entrée de table d'illustration n'a été trouvée.

L'objectif principal de ce cours est d'introduire les théories (en fait, surtout les tenants et aboutissants) derrières les différents paramétrages stratégiques que l'on peut appliquer à une base de données lorsque l'on en est un administrateur¹. L'idée n'est en revanche pas de montrer, sur chacun des « grands moteurs de SGBD », sur quels boutons appuyer pour activer ces paramétrages : la maîtrise des notions théoriques présentées ici doit suffire à faire ces choix quand on y est confronté ensuite...

1.1 Petit retour à la base (de données ! 😊)...

Quelques petits rappels pour commencer : les définitions fondamentales...



Définition : « base de données (BD) »

Une **base de données (BD)** est un ensemble d'informations archivées dans des mémoires accessibles à des ordinateurs en vue de permettre leur traitement², plus ou moins automatisé, par diverses applications prévues à cette fin.

L'intérêt d'une BD est de regrouper les données communes à une application afin³ :

- D'éviter les redondances et les incohérences qu'entraînerait fatalement une approche où les données seraient réparties dans différents fichiers sans connexion entre eux,
- D'offrir des langages de haut niveau pour la définition et la manipulation des données,
- De partager les données entre plusieurs utilisateurs,
- De contrôler l'intégrité, la sécurité et la confidentialité des données,
- D'assurer l'indépendance entre les données et les traitements.

Les 2 « grandes » problématiques usuelles abordées sont donc communément :

- La **modélisation des données** (au travers de formalismes tels que MERISE ou UML par exemple),
- La **manipulation des données** (au travers d'outils, les SGBD, via des langages et/ou des API).

C'est une « classe » de logiciels dédiés, les systèmes de gestion de bases de données (SGBD), qui permettent l'opérationnalisation de la modélisation et de la manipulation des données.



Définition : « système de gestion de bases de données (SGBD) »

Un **système de gestion de bases de données (SGBD)** est un logiciel permettant de gérer une (rarement) ou plusieurs (très majoritairement) bases de données.

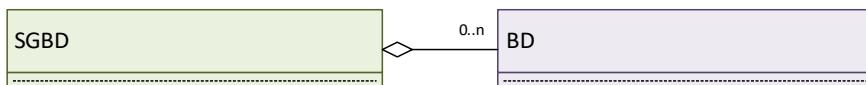


Figure 1. SGBD et BD

¹ Dans les « gros » SGBD (les « petits » n'offrent que rarement ces possibilités de paramétrages, ceux-ci étant alors « imposés » par leur éditeur).

² On entend ici par « traitement » toute opération que l'on peut faire sur une donnée ou un ensemble de données, notamment la vérification, la lecture, l'ajout, la suppression, la modification, l'extraction, ...

³ Ces objectifs ne sont pas tous obligatoirement poursuivis et/ou peuvent, *a contrario*, être complétés par d'autres (*i.e.* cette liste n'est ni exhaustive ni impérative).

Nous allons en parler rapidement ci-après, les BD et SGBD peuvent être caractérisés de plusieurs façons, notamment par...

- *Le paradigme utilisé pour la représentation des données gérées* (cf. §1.2) : il existe en effet plusieurs possibilités de représentation des données (chacune ayant des avantages et des inconvénients) et un SGBD (et, donc, les BD qu'il manipule) est basé sur un (souvent) ou plusieurs (rarement) de ces paradigmes.
- *L'architecture* (cf. §1.3) : celle-ci peut s'entendre de différentes façons...
 - *L'architecture conceptuelle* : souvent basée sur le modèle de l'ANSI/SPARC, elle décrit les différents niveaux de représentation d'une BD (quel que soit son paradigme, d'ailleurs),
 - *L'architecture logique* : elle représente l'organisation interne du moteur du SGBD, organisation usuellement représentée « en couches »,
 - *L'architecture applicative* : elle représente la façon dont peuvent être « répartis » les différents « aspects » d'une application utilisant une ou des BD.

1.2 Paradigmes de gestion de données

Comme pour les langages de programmation, il existe différents paradigmes de « spécification » d'une base de données et, donc, différents paradigmes sur lesquels les SGBD peuvent reposer.

1.2.1 Les principaux paradigmes de gestion de données

Les principaux paradigmes que l'on peut rencontrer sont les suivants :

- *Le paradigme relationnel* : c'est le paradigme « historique », aujourd'hui encore le plus utilisé. Il demande de décrire l'ensemble des données à gérer grâce à des tables liées entre elles. Il permet d'assurer tout un ensemble de propriétés fort intéressantes pour une bonne gestion des données.



En pratique

En relationnel, les données sont représentées sous la forme de relations, éventuellement « liées » entre elles par un mécanisme de « clés étrangères », où chacune de ces relations peut être vue comme une table dont les colonnes sont ses attributs et les lignes sont les n-uplets qu'elle contient à un instant T. Les langages associés sont souvent SQL ou l'un (ou plusieurs) de ses dérivés (par exemple PL/SQL, Transact-SQL, ...).

- *Le paradigme objet* : apparu peu après l'essor des langages de programmation objet, il demande de décrire l'ensemble des données à gérer grâce à des classes (les données en sont des instances) et des collections (d'instances de classes).



En pratique

En objet, les données sont représentées sous la forme de collections d'objets, ces objets étant eux-mêmes instances d'une ou plusieurs classes et pouvant être liés entre eux par certaines de leurs propriétés. Les langages associés sont souvent OQL (lui-même « calqué » sur SQL) ou l'un (ou plusieurs) de ses dérivés.



Remarque

Idéal sur le papier pour travailler avec des langages de programmation objet, le paradigme objet en BD n'a que très peu été utilisé, notamment en raison de ses performances médiocres (sans parler de la quantité de mémoire de travail nécessaire qui est, elle, très importante). Ainsi, malgré un fort engouement au départ, le paradigme objet n'a pas rencontré le succès escompté et est devenu rapidement un marché de niche. Cependant, pour répondre à la problématique de la pérennité des données manipulées par les programmes développés en objet, les éditeurs de SGBD relationnels ont souvent mis en place des mécanismes de « conversion » objet ↔ relationnel (on parle alors parfois de « SGBD mixtes »). L'intérêt est de faciliter la gestion des données depuis des programmes développés en objet (en automatisant plus ou moins la « conversion » vers et depuis le relationnel et en bénéficiant de toute la puissance de la gestion de données relationnelles) malgré quelques inconvénients plus ou moins importants (l'écriture du modèle de conversion peut être fastidieuse et les performances des conversions sont très moyennes).

- *Le paradigme documentaire* : apparu avec l'essor du Web, il demande de décrire les données *via* des modèles documentaires⁴, instanciés dans des documents souvent semi-structurés.



En pratique

Avec ce paradigme, les données sont représentées sous la forme d'informations, plus ou moins structurées, disséminées dans des collections de documents. Les langages descriptifs sous-jacents offrent souvent des mécanismes de liens entre ces données (intra-document et/ou inter-documents). Il est intéressant de noter que, avec ce paradigme, on n'a pas encore réellement à faire à des SGBD (au sens de logiciels couvrant toute la gestion des données) mais plutôt à des ensembles d'outils logiciels souvent organisés en chaînes de traitement. Les langages associés sont souvent XML (et toute la « galaxie » de langages qui entoure XML) et JSON.

- *Le paradigme multidimensionnel* : apparu avec les entrepôts de données puis le *big data*, il décrit avec plusieurs dimensions (*via* des modèles en étoile, en flocon, ...) les données à gérer.



En pratique

Avec une vision multidimensionnelle, les données sont souvent organisées dans des ensembles « amorphes » mais vus *via* des mécanismes comme les cubes de données, les vues en étoile, en flocon, ... Ici, les problématiques « classiques » du relationnel (*i.e.* la non-redondance des données, l'unicité, ...) ne sont pas forcément prégnantes (voire parfois ignorées). Les langages associés sont MDX ou OLAP-SQL, par exemple.

Le choix d'un paradigme, et donc d'une famille de SGBD plutôt que d'une autre, se fait, *grosso modo*, en regard des mêmes considérations que lorsque l'on parle de paradigmes de programmation...

⁴ Cette possibilité reste cependant optionnelle selon la mise en œuvre pratique choisie...

1.2.2 Le paradigme de ce support : le relationnel

Nous nous focaliserons dans ce cours exclusivement sur le paradigme relationnel : bien qu'il cohabite aujourd'hui avec d'autres (notamment les paradigmes documentaire et multidimensionnel), il reste encore très majoritaire.

1.2.2.1 Relations, attributs et n-uplets

En relationnel, une base de données est constituée de **relations**. Chaque relation représente usuellement un concept (livre, étudiant, voiture, ...), en permettant d'en décrire uniformément des « représentants », via une valuation de caractéristiques communes et pertinentes.



Définition : « relation »

Une **relation** est un ensemble de valeurs permettant de décrire de façon homogène des « représentants » d'un concept du monde réel.

En relationnel, une relation est usuellement représentée par une **table** (vision tabulaire) dont les colonnes (appelées « propriétés », « champs » ou, ce que nous ferons dans ce cours, « **attributs** ») déterminent les caractéristiques à valuer pour chaque représentant du concept décrit et dont les lignes (appelées « enregistrements » ou « tuples » ou, ce que nous ferons dans ce cours, « **n-uplets** ») déterminent les représentants du concept décrit. Chaque case de la table contient la **valeur** du représentant de la ligne considérée pour la caractéristique de la colonne considérée.

Attribut			
Prénom	Nom	Âge	...
Jean	PEUPLU	22	...
Jean	BONBEUR	47	...
Alain	VERSE	22	...
Yann	AKEPOURLUI	29	...

**Valeur de cet attribut
pour ce n-uplet**

Figure 2. Concepts basiques du paradigme relationnel

1.2.2.2 Constituants

Les attributs d'une relation peuvent avoir un intérêt à être considérés en « paquets » : c'est le rôle des **constituants**.



Définition : « constituant d'une relation »

Un **constituant** d'une relation est un sous-ensemble (non-vide) des attributs de cette même relation (au moins 1 et au plus la globalité de ces attributs).



Exemple

Si on omet de prendre en compte l'ordre d'apparition des attributs, une relation ayant 4 attributs A, B, C et D possède 15 constituants :

- 4 comprenant 1 seul attribut : {A}, {B}, {C} et {D},
- 6 comprenant 2 attributs : {AB}, {AC}, {AD}, {BC}, {BD} et {CD},
- 4 comprenant 3 attributs : {ABC}, {ABD}, {ACD} et {BCD},
- 1 comprenant les 4 attributs : {ABCD}.

Tout comme un attribut, un constituant est valué : sa valeur est la concaténation (ordonnée) des valeurs des attributs qui le composent pour le n-uplet considéré.



Exemple

Prénom	Nom	Âge	...
Jean	PEUPLU	22	...
Jean	BONBEUR	47	...
Alain	VERSE	22	...
Yann	AKEPOURLUI	29	...
Constituant de valeur			
"YannAKEPOURLUI"			

1.2.2.3 Clés (primaires et étrangères)

Enfin, des concepts de clés permettent :

- Pour les clés primaires : de vérifier la non-redondance des données au sein d'une relation,
- Pour les clés étrangères : de lier les relations entre elles.



Définition : « clé primaire d'une relation »

La **clé primaire** d'une relation est un constituant de cette relation dont la valeur est forcément unique pour chacun de ses n-uplets ; chaque relation possède une et une seule clé primaire.

Ainsi, par définition, deux n-uplets d'une même relation ne peuvent pas avoir la même valeur pour le constituant étant la clé primaire de cette relation : cela permet de vérifier qu'une relation ne contient pas de doublons (sous-entendu « de doublons par rapport à sa clé primaire »).



Exemple (début)

Si le constituant {Prénom, Nom} est la clé primaire des relations ci-dessous, alors :

- La relation suivante est « correcte » étant donné qu'il n'existe aucun doublon dans les valeurs du constituant clé primaire :

Prénom	Nom	Âge	...
Jean	PEUPLU	22	...
Jean	BONBEUR	47	...
Alain	VERSE	22	...
Yann	AKEPOURLUI	29	...



Exemple (fin)

- La relation suivante est « incorrecte » étant donné qu'il existe au moins un doublon dans les valeurs du constituant clé primaire :

Prénom	Nom	Âge	...
Jean	PEUPLU	22	...
Jean	PEUPLU	47	...
Alain	VERSE	22	...
Yann	AKEPOURLUI	29	...



Le corollaire est le suivant : la valeur d'un constituant clé primaire d'une relation permet d'identifier de façon unique chacun de ses n-uplets.



Définition : « clé étrangère d'une relation »

La **clé étrangère** d'une relation R est un constituant de cette relation faisant référence à la clé primaire d'une autre relation S (appartenant forcément à la même base de données) : la valeur du constituant clé étrangère de R est, pour chacun de ses n-uplets, égale à au moins une des valeurs de la clé primaire de la relation S référencée.

Les clés étrangères permettent de relier les différents concepts (donc les différentes relations de la base de données) entre eux (les étudiants avec les formations, les livres avec les emprunteurs, les voitures avec les propriétaires, ...).



Exemple (début)

Soient 2 relations (de la même base de données), l'une décrivant des formations et l'autre décrivant des étudiants ; le constituant {Prénom, Nom} des formations est une clé étrangère faisant référence au constituant clé primaire {Prénom, Nom} des étudiants⁵.

- Les relations suivantes sont « correctes » étant donné que chaque valeur du constituant clé étrangère des formations existe dans la liste des valeurs du constituant clé primaire des étudiants :

Formation	Prénom	Nom	...
L3G MIAGE	Jean	PEUPLU	...
L3G MIAGE	Jean	BONBEUR	...
M1 MIAGE	Alain	VERSE	...
M2 MIAGE	Yann	AKEPOURLUI	...



Prénom	Nom	Âge	...
Jean	PEUPLU	22	...
Jean	BONBEUR	47	...
Alain	VERSE	22	...
Yann	AKEPOURLUI	29	...

⁵ Les noms des attributs des 2 constituants auraient pu ne pas être identiques sans souci, mais pas leur nombre !



Exemple (fin)

- Les relations suivantes sont « incorrectes » étant donné qu'au moins une valeur du constituant clé étrangère des formations n'existe pas dans la liste des valeurs du constituant clé primaire des étudiants :

Formation	Prénom	Nom	...
L3G MIAGE	Jean	PEUPLU	...
L3G MIAGE	Jean	BONBEUR	...
M1 MIAGE	Alain	VERSE	...
M2 MIAGE	Yann	AKEPOURLUI	...

Prénom	Nom	Âge	...
Jean	PEUPLU	22	...
Jean	BONBEUR	47	...
Alain	VERSE	22	...
Yann	AKEPOURLUI	29	...



1.3 Architectures de BD/SGBD

Comme souvent en informatique, le terme « architecture » peut être décliné à plusieurs niveaux. Dans le monde des BD/SGBD, on parle notamment d'architecture conceptuelle (cf. §1.3.1) ainsi que d'architecture logique (cf. §1.3.2).

1.3.1 Architecture(s) conceptuelle(s)

Il existe plusieurs architectures conceptuelles de BD. Cependant, en 1975, le comité SPARC de l'ANSI a proposé une architecture à 3 niveaux qui est devenue « classique » (cf. Figure 3). Chacun de ces niveaux, que l'on appelle des « schémas », correspond à une « vision », une « représentation », d'une BD. Selon cette architecture ANSI/SPARC, il existe ainsi, pour toute base de données :

- *Un schéma interne* (cf. §1.3.1.1) : il a trait à l'organisation des données en mémoire de stockage,
- *Un schéma conceptuel* (cf. §1.3.1.2) : il a trait au modèle des données,
- *Un ou plusieurs schémas externes* (cf. §1.3.1.3) : ils ont trait à la présentation des données.

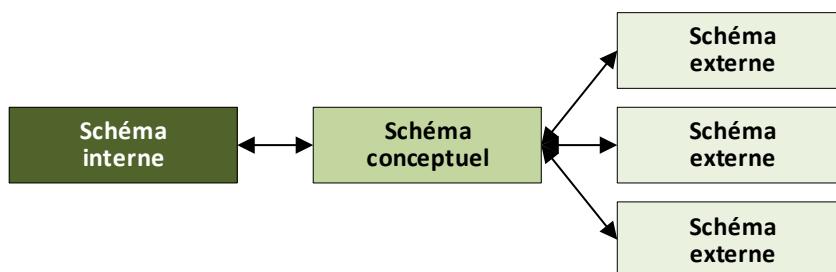


Figure 3. Architecture ANSI/SPARC à 3 niveaux



Remarque

Chacun de ces niveaux peut usuellement être « exprimé » de différentes manières. L'expression choisie pour chaque niveau est, bien sûr, dépendante (plus ou moins selon le niveau considéré) du paradigme sélectionné.

1.3.1.1 Schéma interne (ou « schéma physique »)



Définition : « schéma interne » ou « schéma physique »

Le **schéma interne** décrit l'organisation des données en mémoire de stockage. Il existe plusieurs façons de réaliser cette organisation : le schéma physique d'une BD montre l'organisation physique choisie pour cette BD.

L'organisation choisie doit permettre d'accéder le plus rapidement possible à un ensemble de données vérifiant certaines conditions et de créer/modifier/supprimer des données avec une réorganisation minimale (*i.e.* la moins coûteuse) et une utilisation optimale (*i.e.* la plus petite) de la place disponible (notamment afin de limiter les transferts entre la mémoire de stockage et la mémoire de travail).



En pratique

Il existe plusieurs organisations de données à ce niveau. Les plus courantes sont :

- Les organisations séquentielles,
- Les organisations multi-listes,
- Les organisations indexées (plusieurs types d'index existent : index arborescents, index à accès par adressage calculé, ...).

Nous le verrons (cf. §6), l'organisation des données à ce niveau repose sur différents choix stratégiques à réaliser, notamment :

- Le format de stockage des valeurs d'attributs (variable ou fixe),
- La gestion du répertoire des déplacements situé à la fin des « pages de stockage » (dynamique ou statique) des données,
- Le mode d'adressage (direct, indirect ou mixte).



Attention

Il est extrêmement complexe (et coûteux) de changer d'organisation physique une fois que l'on en a choisi une pour une BD (et encore, quand cette modification est possible, ce qui n'est pas toujours le cas !).

1.3.1.2 Schéma conceptuel



Définition : « schéma conceptuel »

Ce niveau décrit la façon dont « le monde réel est représenté dans la BD ». Il s'agit donc d'un niveau de modélisation. Ainsi, le **schéma conceptuel** prend la forme d'un modèle conceptuel (dont la formalisation dépend bien sûr du paradigme choisi, cf. §1.2). La BD est une instance de ce modèle conceptuel (dont les données sont stockées au niveau physique en fonction du schéma interne, cf. §1.3.1.1).

Les principaux concepts usuels sont les suivants :

- *Entité* : une personne, un livre, ...
- *Propriété (attribut)* : titre d'un livre, adresse postale d'une personne, ...
- *Association* : personne auteur d'un livre, ...
- *Agrégation* : une adresse composée d'une rue et d'un code postal, ...
- *Collection* : un ensemble de personnes, une liste de livres, ...

On parle souvent de « *générations de modèles conceptuels* » (on peut grossièrement lier ces générations au différents paradigmes). Les principaux modèles conceptuels connus sont :

- *1^{ère} génération :*
 - Modèle hiérarchique (IMS d'IBM),
 - Modèle réseau (DBTG CODASYL).
- *2^{ème} génération :*
 - Modèle relationnel,
 - Modèle entité-association.
- *3^{ème} génération :*
 - Modèle orienté objet,
 - Modèle objet/relationnel.
- *4^{ème} génération :*
 - Modèle semi-structuré « puissant » (XML),
 - Modèle semi-structuré « léger » (JSON).



Exemple

Soit une BD contenant les livres d'une bibliothèque, leurs auteurs et les abonnés à la bibliothèque. On suppose qu'un livre est identifié par sa cote et qu'un auteur et un abonné sont identifiés par leur nom. La formalisation de cette hypothèse constitue le schéma conceptuel de la BD.

Les principaux modèles sont introduits ci-après via une illustration de leurs principaux concepts...



Remarque

Le schéma conceptuel est celui qui est le plus « visuellement » lié à un paradigme donné...

1.3.1.2.1 Dans le cadre du paradigme relationnel

Imaginons que l'on souhaite représenter une BD relative à des personnes décrites par leur nom, leur âge et leur situation matrimoniale. Avec un modèle relationnel, une BD est décrite :

- *De façon « intensive », via le schéma conceptuel :* on parle alors de l'**intension**⁶ de la BD (cf. Figure 4). Elle est alors modélisée par un ensemble de **relations**, chacune décrite par son schéma et ce **schéma de la relation** est lui-même constitué du **nom de la relation** et de la liste de ses attributs, chacun de ces **attributs** étant lui-même décrit par le **nom de l'attribut** considéré et son **domaine**, le domaine étant l'ensemble des valeurs que l'attribut associé peut prendre⁷.

⁶ On utilise bien dans ce cadre le mot « intension », qui signifie « compréhension, entendement », et non « intention », qui signifie « idée, pensée, motif ».

⁷ Le domaine de valeurs d'un attribut peut être décrit intensionnellement (*i.e.* par un type de données par exemple) ou extensionnellement (*i.e.* avec la liste exhaustive des valeurs autorisées).

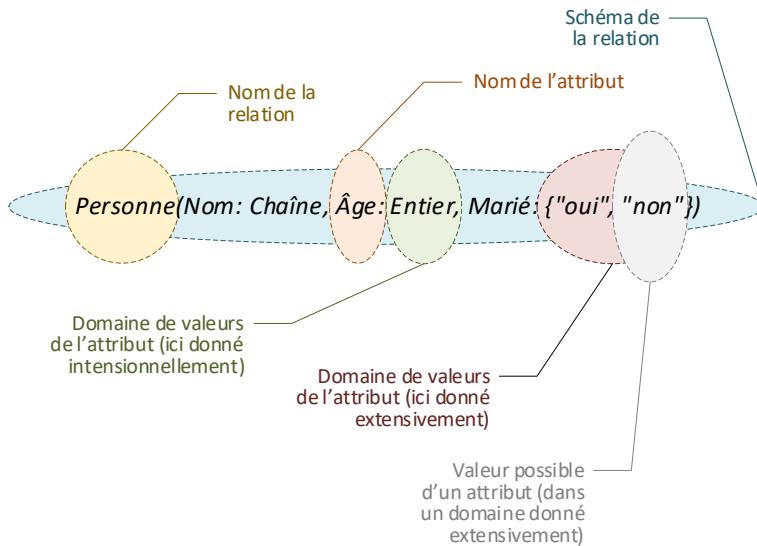


Figure 4. Schéma conceptuel relationnel



Définition : « intension »

On parle d'**intension** d'une BD dans le cadre du paradigme relationnel notamment⁸. Il s'agit de son schéma conceptuel, i.e. de son modèle conceptuel. L'intension d'une BD est relativement pérenne (elle n'est normalement que peu amenée à évoluer).

- *De façon « extensive », via le schéma interne* : on parle de l'**extension** d'une BD (cf. Figure 5), qui est en fait l'instance, à un moment T, de l'intension de la BD. C'est la liste complète de l'ensemble des données contenues à cet instant dans la BD, groupées par relation. On trouve donc ici des **extensions des relations** (i.e. la liste complète des données de la BD, relation par relation), l'extension d'une relation étant constituée de **n-uplets** (ou « tuples » ou « enregistrements »), chaque n-uplet indiquant la **valeur d'attribut** que possède chaque attribut de cette relation-là pour lui.

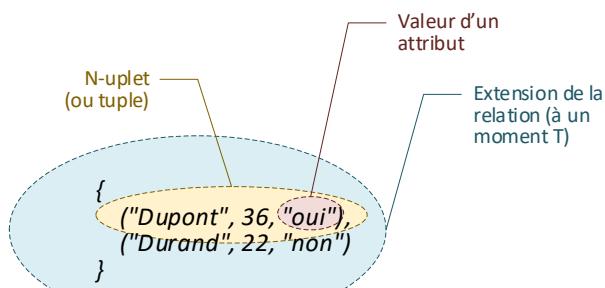


Figure 5. Instance de schéma conceptuel relationnel

⁸ On peut (rarement) rencontrer aussi ce terme dans le cadre d'autres paradigmes.



Définition : « extension »

On parle d'**extension** d'une BD dans le cadre du paradigme relationnel notamment⁸. Il s'agit d'une instance de son schéma conceptuel à un moment T, *i.e.* de l'ensemble des données qu'elle contient à cet instant-là, présentées en fonction du modèle conceptuel tel qu'il a été défini. L'extension d'une BD est usuellement modifiée extrêmement souvent (dès que l'on ajoute/modifie/supprime une donnée).



Exemple

Soit une BD décrivant (sommairement !) une bibliothèque. On décide de décrire notamment des livres (identifiés de façon unique par une cote et ayant un titre), des auteurs (identifiés de façon unique par un nom⁹ et le titre du¹⁰ livre qu'ils ont écrit) et des abonnés à la bibliothèque (identifiés de façon unique par un nom¹¹, un prénom et une année de naissance). Avec un modèle relationnel, on peut décrire le schéma conceptuel de cette BD comme suit, par l'intension de la BD :

Livre	(<u>Cote: Chaîne</u> , <u>Titre: Chaîne</u>)
Auteur	(<u>Nom: Chaîne</u> , <u>Cote: Chaîne</u>)
Abonné	(<u>Nom: Chaîne</u> , <u>Prénom: Chaîne</u> , <u>Année_Naissance: Entier</u>)

Figure 6. Exemple de schéma conceptuel relationnel

Ce schéma conceptuel peut alors par exemple être instancié de la façon suivante (extrait de l'extension de la BD, telle qu'elle existe à un moment T) :

Livre		Auteur	
Cote	Titre	Nom	Cote
BD/46	Les BD en BD	Dupont	BD/46
...	...	Durand	BD/46
	

Abonné		
Nom	Prénom	Année_Naissance
Martin	Jean	1960
Michel	Pierre	1953
...

Figure 7. Exemple d'instance de schéma conceptuel relationnel



Remarque

Cette « vision tabulaire » de l'extension d'une relation dans le paradigme relationnel n'est en fait pas la seule : avec ce paradigme, la même extension peut également être appréhendé avec une « vision assertionnelle », vision étroitement liée à l'algèbre relationnelle notamment.

⁹ Pour simplifier, on suppose qu'on ne peut pas avoir d'homonymes parmi les auteurs.

¹⁰ Pour simplifier, on suppose qu'un auteur n'écrit qu'un seul livre (en revanche, un même livre peut avoir été écrit par plusieurs auteurs).

¹¹ Pour simplifier, on suppose qu'on ne peut avoir d'homonymes parmi les abonnés.

1.3.1.2.2 Dans le cadre du paradigme objet

Le paradigme relationnel a historiquement été majoritairement le plus utilisé depuis l'essor des BD/SGBD. Cependant, lorsque les langages de programmation orientés objet ont pris de plus en plus d'importance dans le développement d'applications, la sauvegarde de données a vite posé des problèmes (intuitivement, on voit bien qu'une structure relationnelle n'a que peu de choses à voir avec une structuration objet). C'est pourquoi le paradigme objet, sous-tendant la programmation objet, a été étendu aux BD. Avec ce paradigme, on manipule des concepts objet classiques tels que les classes (assimilables aux relations), les propriétés (assimilables aux attributs), les collections (assimilables aux extensions de relations), ...

Exemple

Toujours avec un exemple de bibliothèque, mais plus évolué¹², on a par exemple ici, au niveau du schéma conceptuel de la BD :

```

classe Livre
  propriété Cote : Chaîne
  propriété Titre: Chaîne
  propriété Auteurs : collection(Personne)
classe Personne
  propriété Nom : Chaîne
  propriété Prénom : Chaîne
  propriété AnnéeNaissance : Entier
  méthode Age: Entier
    {annéeCourante - self->Année_Naissance}

  Livres : collection(Livre)
  Auteurs : collection(Personne)
  Abonnés : collections(Personne)

```

Figure 8. Exemple de schéma conceptuel orienté objet



Ce schéma conceptuel peut alors par exemple être instancié de la façon suivante :

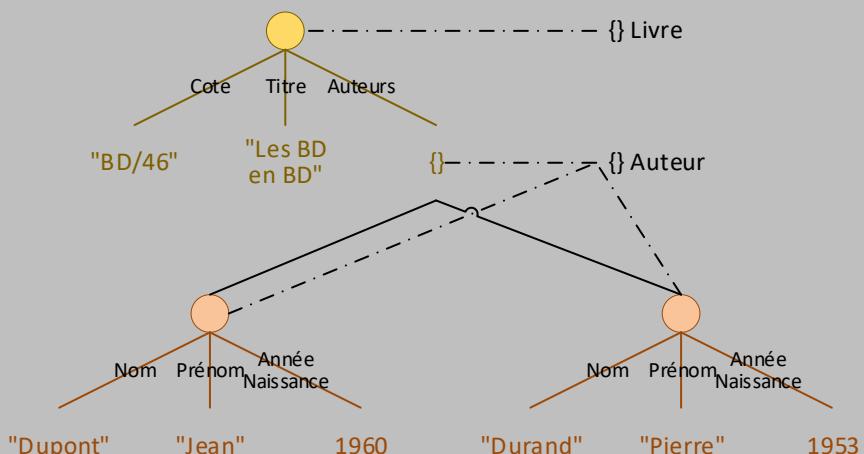


Figure 9. Exemple d'instance de schéma conceptuel orienté objet

¹² On introduit en effet ici de nouveaux aspects (par exemple la possibilité pour un auteur d'écrire plusieurs livres).

Remarque

Tout comme dans le paradigme relationnel une extension peut adopter une « vision tabulaire » ou une « vision assertionnelle », dans le paradigme objet une extension peut adopter une « vision arborescente » (comme dans l'exemple précédent) ou une « vision linéaire » ou « vision référentielle » (comme ci-dessous) :

```
P1: [Nom = "Dupont", Prénom = "Jean",
      AnnéeNaissance = 1960]
P2: [Nom = "Durand", Prénom = "Pierre",
      AnnéeNaissance = 1953]
L1: [Cote = "BD/46", Titre = "Les BD en BD",
      Auteurs = {P1, P2}]

Livres = {L1}
Personnes = {P1, P2}
```

Figure 10. Exemple d'instance de schéma conceptuel orienté objet (autre vision)

1.3.1.2.3 Dans le cadre du paradigme documentaire

Avec l'essor du Web, la manipulation de données a adopté un « nouveau » paradigme : le paradigme documentaire (notamment mis en œuvre *via* les technologies XML). On peut donc baser un schéma conceptuel sur ce paradigme, notamment avec des concepts d'éléments et d'attributs.

Remarque

Nous ne reviendrons bien sûr pas sur les tenants et aboutissants de ce paradigme, largement présenté l'année dernière et que vous maîtrisez donc maintenant parfaitement ! 😊

Dans ce cadre, le schéma conceptuel est décrit par un modèle de documents (DTD, XML Schema, ...) et son instance par un document (XML) valide par rapport à ce modèle.

Exemple (début)

Encore et toujours avec un exemple de description d'une bibliothèque, son intension (*i.e.* son schéma conceptuel) pourrait être le suivant (ici, une DTD) :

```
<!ELEMENT bibliothèque (livres, personnes)>
<!ELEMENT livres (livre*)>
<!ELEMENT livre (cote, titre, auteur*)>
<!ELEMENT cote (#PCDATA)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur EMPTY>
<!ATTLIST auteur ref IDREF>
<!ELEMENT personnes (personne*)>
<!ELEMENT personne (nom, prénom, année_naissance)>
<!ATTLIST personne id ID>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prénom (#PCDATA)>
<!ELEMENT année_naissance (#PCDATA)>
```

Figure 11. Exemple de schéma conceptuel documentaire



Exemple (fin)

Ce schéma conceptuel peut alors par exemple être instancié de la façon suivante :

```
<bibliothèque>
  <livres>
    <livre>
      <cote>BD/46</cote>
      <titre>Les BD en BD</titre>
      <auteurs>
        <auteur ref="P1"/>
        <auteur ref="P2"/>
      </auteurs>
    </livre>
  </livres>
  <personnes>
    <personne id="P1">
      <nom>Dupont</nom>
      <prenom>Jean</prenom>
      <année_naissance>1960</année_naissance>
    </personne>
    <personne id="P2">
      <nom>Durand</nom>
      <prenom>Pierre</prenom>
      <année_naissance>1953</année_naissance>
    </personne>
  </personnes>
</bibliothèque>
```

Figure 12. Exemple d'instance de schéma conceptuel documentaire

1.3.1.3 Schéma externe

Un schéma externe représente la façon dont un utilisateur final, ou une application, voit la partie de la BD qui le concerne. Il existe en général plusieurs schémas externes pour une même BD : le schéma conceptuel d'une BD pouvant être complexe, les schémas externes donnent aux utilisateurs une vision plus simple et partielle du schéma conceptuel. Les schémas externes permettent aussi de protéger la BD contre des manipulations incorrectes ou non autorisées, en cachant certaines données à certains utilisateurs. Au niveau externe, l'accès à la BD est réalisé par des langages de haut niveau (menus, formulaires, langage naturel, ...).



Définition : « schéma externe »

Un (puisque'il y en a souvent plusieurs pour une même BD) **schéma externe** d'une BD est un mode de présentation de tout ou partie des données qu'elle contient. Un schéma externe donné est souvent dédié à une catégorie d'utilisateurs, éventuellement pour une catégorie particulière de traitements, afin de ne présenter que les données nécessaires à ces traitements et auxquelles l'utilisateur a le droit d'accéder.

1.3.1.4 Pourquoi ces 3 niveaux ? L'indépendance données/traitements

L'indépendance données/traitements est indispensable pour pouvoir faire évoluer facilement l'organisation physique ou logique d'une BD ou bien l'architecture matérielle de la machine sur laquelle tourne le SGBD qui la gère. Cette indépendance est l'un des objectifs majeurs de l'utilisation d'un SGBD. Si elle est atteinte, l'indépendance données/traitements permet :

- De modifier l'organisation physique ou l'indexation (par exemple ajouter un index pour un accès plus rapide) sans modifier le schéma conceptuel ni les applications,
- De modifier le schéma conceptuel (par exemple ajouter un nouveau type d'entité ou d'association) sans se soucier de l'organisation physique (gérée de façon transparente).

On parle aussi d'**indépendance logique** et d'**indépendance physique**.



Pour en savoir plus

On peut se documenter aisément sur le Web sur cette forme d'indépendance. Par exemple¹³ :

« *L'architecture à trois niveaux, définie par le standard ANSI/SPARC, permet d'avoir une indépendance entre les données et les traitements. D'une manière générale, un SGBD doit avoir les caractéristiques suivantes :*

- Indépendance physique : *le niveau physique peut être modifié indépendamment du niveau conceptuel. Cela signifie que tous les aspects matériels de la base de données n'apparaissent pas pour l'utilisateur, il s'agit simplement d'une structure transparente de représentation des informations.*
- Indépendance logique : *le niveau conceptuel doit pouvoir être modifié sans remettre en cause le niveau physique, c'est-à-dire que l'administrateur de la base doit pouvoir la faire évoluer sans que cela gêne les utilisateurs.*
- Manipulabilité : *des personnes ne connaissant pas la base de données doivent être capables de décrire leur requête sans faire référence à des éléments techniques de la base de données.* »

1.3.2 Architecture logique

Un SGBD est un logiciel souvent vu en « couches », chaque couche étant un « fragment » de ce logiciel¹⁴. Ces couches communiquent entre elles et font l'interface entre les utilisateurs (humains ou programmes applicatifs) du SGBD et les données contenues dans les BD manipulées et enregistrées physiquement sur une mémoire de stockage. Cette architecture logique en couches permet une plus grande « souplesse » (i.e. modularité) dans l'implémentation du SGBD et dans les services qu'il est capable de fournir.

¹³ Source : <https://www.commentcamarche.net/contents/102-les-niveaux-de-donnees>

¹⁴ On parle souvent de « service » même si, dans les faits, très souvent ce ne sont pas des « services » au sens informatique moderne du terme.

Vue très grossièrement (cf. Figure 13), l'architecture logique type d'un SGBD est la suivante (les couches sont abordées de « haut », *i.e.* des utilisateurs, en « bas », *i.e.* aux données enregistrées en mémoire de stockage) :

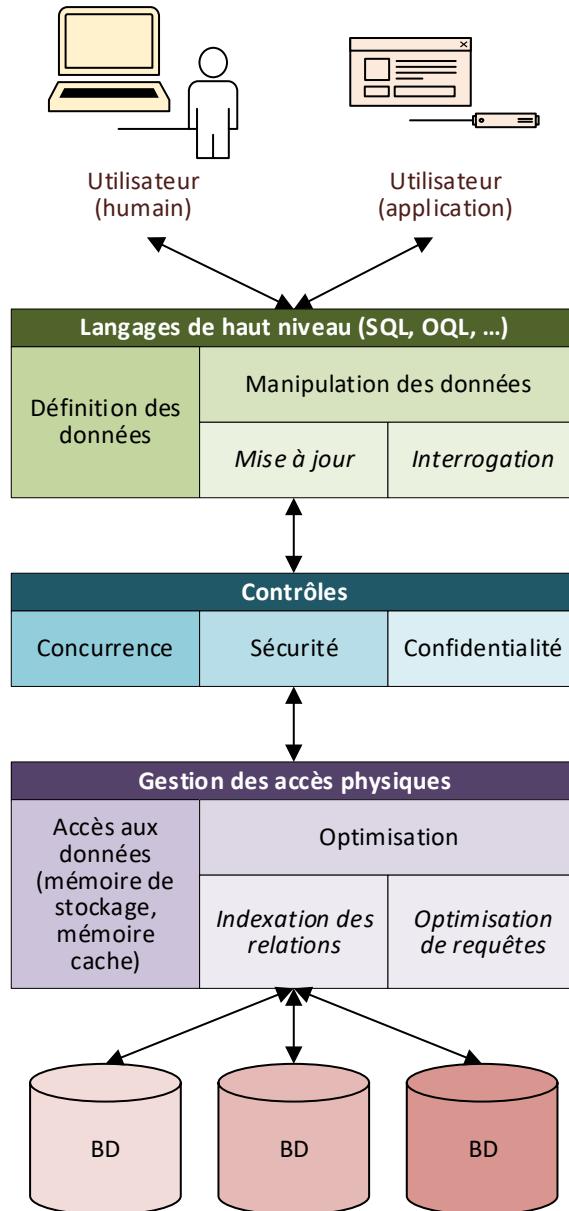


Figure 13. Architecture logique type d'un SGBD

1.4 Pourquoi un SGBD ?

Un SGBD, nous l'avons vu (cf. §1.1), est un logiciel permettant fondamentalement de « gérer » une ou plusieurs BD. Cela sous-entend couramment les 2 fonctions suivantes :

- **Permettre la définition des données** : notamment *via* la définition de tables, avec leurs attributs et les domaines de valeurs de ceux-ci, ainsi que *via* la définition de clés (primaires et étrangères), ce qui sous-entend que le SGBD puisse assurer en permanence le respect des contraintes liées aux clés ainsi définies.
- **Permettre la manipulation des données** : notamment *via* le remplissage (ajout, suppression, modification de données) des tables des bases gérées ainsi que *via* l'écriture (puis l'exécution, bien sûr) de requêtes.

En outre, et on l'oublie trop souvent, un SGBD doit impérativement assurer 2 autres fonctions :

- **Garantir la cohérence des données** : un SGBD doit en effet assurer en permanence que les données gérées respectent tout un ensemble de contraintes, au premier rang desquelles celles qui sont liées à leur définition.
- **Offrir des performances optimales** : sans quoi, le SGBD perdrat tout intérêt pratique ☺

Ce cours se focalise sur ces 2 dernières fonctions (cohérence et optimalité) au travers de la présentation, pour chaque problématique abordée, des catégories de stratégies pouvant être proposées afin de permettre leur identification et, après analyse du contexte, de vous guider vers les choix pertinents. Ainsi, un SGBD implémente des mécanismes dédiés à ces problématiques. On parle souvent de « gestionnaires », voire de « moteurs », et ceux-ci « communiquent » souvent les uns avec les autres. On peut replacer ces mécanismes, qui communiquent entre eux, par rapport aux niveaux de l'architecture logique du SGBD (cf. Figure 14).



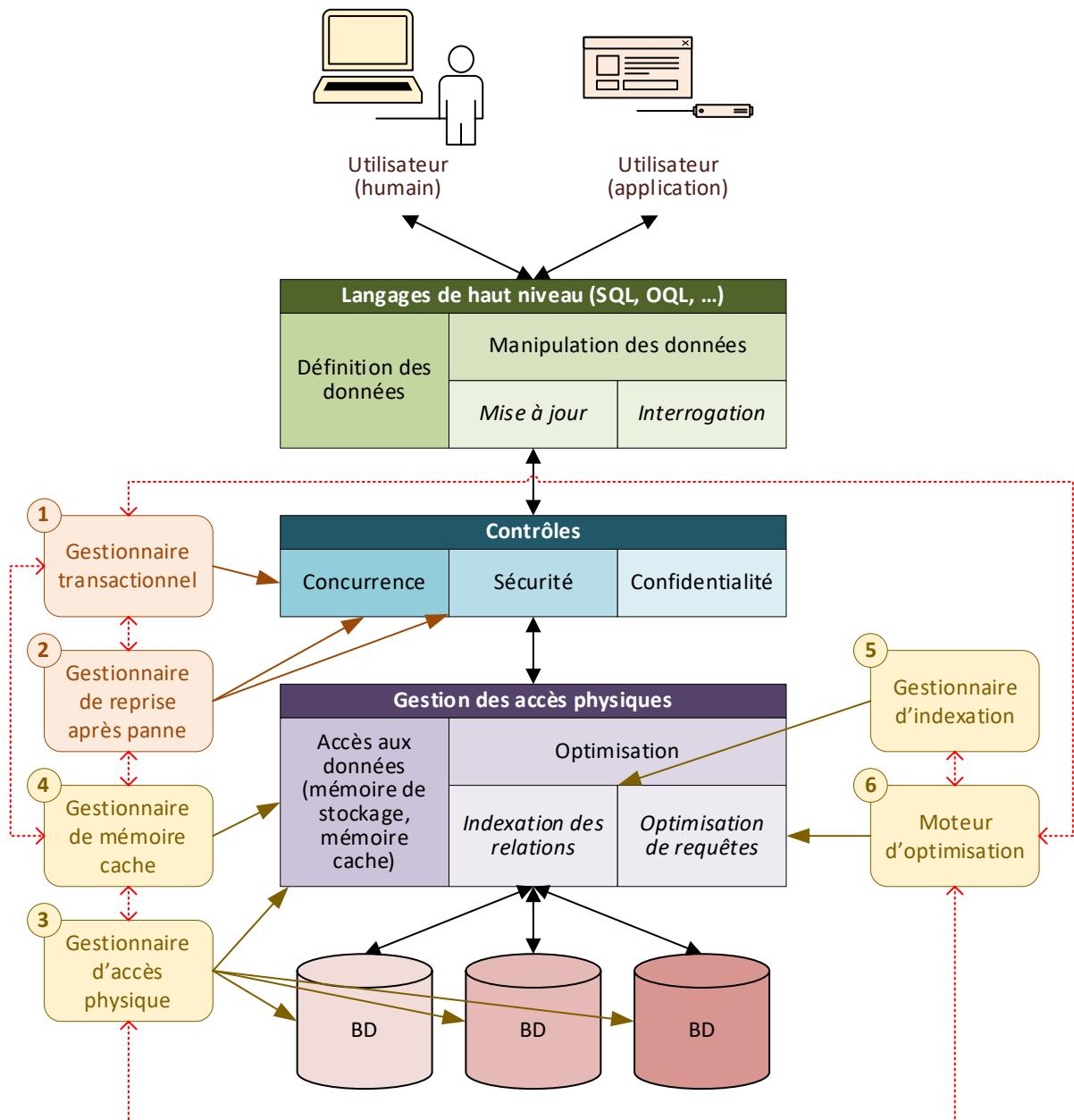
Remarque

Bien sûr, un SGBD est également composé de tout un panel d'autres gestionnaires, services, moteurs, ..., mais que nous n'aborderons pas ici.

Décris sommairement, ces mécanismes sont les suivants :

- **Au niveau du contrôle des données :**
 - *Le gestionnaire transactionnel* : c'est lui qui assure le support de la notion de « transaction » et, donc, les propriétés ACID liées ; il participe au respect de la cohérence de la BD,
 - *Le gestionnaire de reprise après panne* : il assure la remise dans un état cohérent d'une BD¹⁵ après une panne du SGBD,
- **Au niveau de l'optimisation des performances :**
 - *Le gestionnaire d'accès physique* : c'est lui qui assure l'accès aux données stockées sur la mémoire de stockage ; il participe, en conjonction avec un ou plusieurs langages de haut niveau (SQL, OQL, ...), à l'indépendance données/traitements,
 - *Le gestionnaire de mémoire cache* : il participe également à l'accès aux données en essayant de maximiser les performances de ces accès (*via* une mémoire cache),
 - *Le gestionnaire d'indexation* : lui aussi participe à la maximisation des performances d'accès aux données (*via* la définition et la manipulation d'un ou plusieurs index),
 - *Le moteur d'optimisation* : il assure la maximisation des performances d'exécution d'une requête.

¹⁵ Quitte à « perdre » des ajouts/modifications/suppressions ! En effet, la cohérence de la BD est bien plus importante que la pérennité de certaines opérations (il suffira de relancer les traitements « perdus »).



Ces mécanismes participent ainsi de près ou de loin à « la bonne tenue » d'une BD par un SGBD.

Mécanismes (services/moteurs/gestionnaires)	Isolation physique ¹⁶	Contrôle des données ¹⁷				Propriétés ACID ¹⁷				Optimisation ¹⁸
		Cohérence	Concurrence	Sécurité	Confidentialité	Atomicité	Cohérence	Isolation	Durabilité	
1. Gestionnaire transactionnel		X	X	X		X	X	X		
2. Gestionnaire de reprise après panne		X		X			X		X	
3. Gestionnaire d'accès physique	X									X
4. Gestionnaire de mémoire cache				X						X
5. Gestionnaire d'indexation										X
6. Moteur d'optimisation										X

Tableau 1. Apports des principaux mécanismes d'un SGBD



Remarque

Notez que la confidentialité n'est adressée par aucun des mécanismes présentés ici. Elle est en fait assurée par un mécanisme de gestion des droits d'accès (plus ou moins évolué selon les SGBD) mais ce cours n'aborde pas ce point.

1.5 OS et SGBD, même combat ?

La question ci-dessus peut paraître être un non-sens mais en fait pas du tout : de plus en plus de technologies des SGBD se retrouvent mises en œuvre au sein des OS¹⁹ (parfois l'inverse est aussi vrai mais très nettement plus rarement), notamment :

- Les mécanismes de mise en mémoire cache²⁰,
- Les mécanismes de pagination pour la gestion de la mémoire,
- Les mécanismes d'indexation (pour accélérer les recherches et accès aux fichiers notamment),
- Des mécanismes d'optimisation,
- ...

¹⁶ Cf. §1.3.1.4.

¹⁷ Ces notions seront détaillées dans le Livre 1.

¹⁸ Les notions sous-jacentes seront détaillées dans le Livre 2.

¹⁹ Systèmes d'exploitation (*operating systems* en anglais).

²⁰ Utilisés même au niveau physique, *i.e.* dans l'électronique des ordinateurs (processus, disques, ...).

Stratégies de gestion de données relationnelles

Livre 1 : assurer le contrôle des données

2 Introduction au contrôle des données

SOMMAIRE DÉTAILLÉ DU CHAPITRE 2

2.1	Le souci du contrôle des données.....	25
2.2	Vers le concept de « transaction » et les propriétés ACID	26
2.3	Assurer la pérennité des données	26

FIGURES DU CHAPITRE 2

Figure 18. Exemple de transaction 26

TABLEAUX DU CHAPITRE 2

Aucune entrée de table d'illustration n'a été trouvée.

ÉQUATIONS DU CHAPITRE 2

Aucune entrée de table d'illustration n'a été trouvée.

Nous avons déjà vu que le SGBD doit assurer l'indépendance données/traitements (cf. §1.3.1.4) ce qui apporte une indépendance physique, une indépendance logique et aussi, de façon corollaire, une certaine maniabilité des données (basée sur un principe d'abstraction mis en œuvre par la modélisation des données).

2.1 Le souci du contrôle des données

D'autres propriétés, liées au « contrôle des données », doivent néanmoins être assurées. Il existe 4 types classiques de contrôles des données :

- *Cohérence* : les données stockées dans une BD doivent respecter un certain nombre de contraintes (quand c'est le cas on dit que « la BD est dans un état cohérent »). Ces contraintes peuvent être exprimées de façon déclarative au niveau du schéma conceptuel ou bien être associées aux opérations de mise à jour. Un SGBD doit assurer que les contraintes sont respectées.

Question

De quels types de contraintes parle-t-on dans ce contexte ?

Réponse

On en distingue souvent 3 types :

- *Les contraintes de types* : on l'a vu, les attributs/propriétés sont typés (plus ou moins fortement). Le SGBD doit assurer qu'à tout instant toute valeur assignée à tout attribut existe bien dans le domaine de valeurs associé à cet attribut.
- *Les contraintes d'intégrité* : ces contraintes regroupent aussi bien les mécanismes assimilables à l'unicité de valeur et/ou les clés primaires que les mécanismes assimilables à des liens internes ou externes (clés étrangères par exemple). Le SGBD doit assurer qu'à tout instant chacune de ces contraintes d'intégrité est respectée dans une BD.
- *Les contraintes événementielles* : elles sont associées, comme leur nom l'indique, à un événement mais aussi à un ensemble de traitements. Le SGBD doit assurer que dès qu'un événement visé par une contrainte événementielle survient alors les traitements associés à cette contrainte sont exécutés.

- *Concurrence* : en général plusieurs utilisateurs « se partagent » la même BD. Plusieurs traitements peuvent donc s'exécuter en même temps. Un SGBD doit assurer que les éventuels conflits entre ces traitements ne mettent pas la BD dans un état incohérent.
- *Sécurité* : après une panne, qu'elle soit d'origine logicielle ou matérielle, un SGBD doit être capable de restaurer la BD dans un état cohérent, si possible le même ou le plus proche de celui précédent la panne.
- *Confidentialité* : un SGBD doit permettre d'interdire à certaines personnes d'accéder à la BD ou encore de réaliser certaines opérations sur une partie ou sur toute la BD.

2.2 Vers le concept de « transaction » et les propriétés ACID

Plusieurs mécanismes sont mis en œuvre au sein des SGBD pour assurer ces 4 problématiques de contrôle des données. Ces mécanismes sont tous basés sur différents concepts dont le plus prégnant est celui de transaction.



Définition : « transaction »

Une **transaction** est un ensemble d'opérations élémentaires considéré comme « un tout ». Une transaction peut donc être considérée comme une opération complexe faisant passer la BD manipulée d'un état cohérent à un autre état cohérent une fois que l'ensemble des opérations élémentaires composant la transaction ont été exécutées. Ainsi, pour un SGBD, la transaction est l'unité de traitement.

L'idée derrière cette notion est la suivante : bien que composite, un SGBD doit assurer qu'une transaction est exécutée complètement et de façon pérenne ou alors pas du tout (quitte à annuler certaines des opérations élémentaires de la transaction déjà effectuées).



Exemple

Un exemple classique de transaction est l'opération qui transfère une somme S d'un compte bancaire A à un compte bancaire B :

```
début transaction
    solde(A) = solde(A) - S
    solde(B) = solde (B) + S
fin transaction
```

Figure 15. Exemple de transaction

Il est clair que cette opération ne doit pas être interrompue entre le débit du compte bancaire A et le crédit du compte bancaire B .

Ce concept de transaction permet la mise en œuvre de mécanismes garantissant de nouvelles propriétés appelées « propriétés ACID » (en raison de leurs initiales). Elles sont au nombre de 4 :

- **Atomicité** : soit toutes les modifications effectuées par une transaction sont enregistrées dans la BD, soit aucune ne l'est. Ainsi :
 - Si une transaction est confirmée (`commit`), toutes les modifications qu'elle a effectuées sont enregistrées dans la BD et rendues visibles aux autres utilisateurs,
 - Si une transaction est interrompue (`rollback` ou panne), alors aucune de ses modifications n'est enregistrée dans la BD (les modifications déjà effectuées doivent être annulées).
- **Cohérence** : une transaction fait passer une BD d'un état cohérent à un autre état cohérent.
- **Isolation** : une transaction se déroule sans être perturbée par les transactions concurrentes : tout se passe comme si elle se déroulait seule.
- **Durabilité** : une fois qu'une transaction a été confirmée (`commit`), le SGBD garantit qu'aucune modification qu'elle a effectuée ne sera perdue, quels que soient les accidents qui surviendront : interruption, panne du système d'exploitation, « crash » disque, ...

2.3 Assurer la pérennité des données

Contrôler les données nécessite également d'assurer leur pérennité : il s'agit de garantir au maximum que les données stockées sont cohérentes et ce de façon durable, y compris en cas de panne du SGBD.

3 Gestionnaire transactionnel

SOMMAIRE DÉTAILLÉ DU CHAPITRE 3

3.1	Notion de transaction	29
3.2	Propagation des modifications	32
3.2.1	Mode de mise à jour immédiate	32
3.2.2	Mode de mise à jour différée	33
3.3	Problèmes dus à la concurrence	34
3.3.1	Perte de mise à jour	35
3.3.2	Lecture impropre (données incohérentes).....	35
3.3.3	Lecture impropre (données non confirmées).....	36
3.3.4	Lecture non reproductible.....	37
3.3.5	Comment éviter ces problèmes d'accès concurrentiels ?	38
3.4	Contrôle de concurrence	40
3.4.1	Sérialisabilité d'une exécution concurrente de transactions.....	40
3.4.1.1	Équivalence d'exécutions de transactions.....	40
3.4.1.2	Sérialisabilité d'une exécution concurrente de transactions.....	42
3.4.1.3	Conditions de sérialisabilité d'une exécution concurrente	43
3.4.1.4	Détection de la non-sérialisabilité : graphe de précédence.....	48
3.4.1.5	Traitements en cas de non-sérialisabilité	54
3.4.2	Verrouillage : mieux vaut prévenir que guérir.....	55
3.4.2.1	Modes de verrouillage.....	55
3.4.2.2	Verrouillage à deux phases.....	58
3.4.2.2.1	Technique du verrouillage à deux phases	59
3.4.2.2.2	Verrouillage à deux phases et sérialisabilité	60
3.4.2.3	Interblocage	63
3.4.2.3.1	Détection d'un interblocage : graphe d'attente.....	64
3.4.2.3.2	Traitements de l'interblocage	65
3.4.2.4	Reclassement d'un verrou	66
3.4.2.5	Verrouillage à granularité multiple	68
3.5	Paramètres du gestionnaire transactionnel.....	74

TABLEAUX DU CHAPITRE 3

Tableau 2.	Événements transactionnels basiques	31
Tableau 3.	Nature des opérations transactionnelles selon le mode de mise à jour	43
Tableau 4.	Permutation de 2 opérations op_{NOBD} sans accès BD	44
Tableau 5.	Permutation d'une opération op_{NOBD} sans accès BD et d'une lecture op_{lect}	44
Tableau 6.	Permutation d'une opération op_{NOBD} sans accès BD et d'une écriture op_{Ecrit}	45
Tableau 7.	Permutation de 2 lectures op_{lect} de 2 données différentes	45
Tableau 8.	Permutation de 2 lectures op_{lect} d'une même donnée	45
Tableau 9.	Permutation d'une lecture op_{lect} et d'une écriture op_{Ecrit} de 2 données différentes	45
Tableau 10.	Permutation d'une lecture op_{lect} et d'une écriture op_{Ecrit} d'une même donnée	46
Tableau 11.	Permutation de 2 écritures op_{Ecrit} de 2 données différentes	46
Tableau 12.	Permutation de 2 écritures op_{Ecrit} d'une même donnée	46
Tableau 13.	Synthèse des conflictualités entre types d'opérations transactionnelles	46
Tableau 14.	Matrice de compatibilité des verrous en mode S et X	55
Tableau 15.	Événements transactionnels de verrouillage/déverrouillage	56
Tableau 16.	Avantages et inconvénients des stratégies de traitement de l'interblocage	66
Tableau 17.	Événements transactionnels de reclassement	67
Tableau 18.	Matrice de compatibilité des verrous « classiques » ET intentionnels.....	71
Tableau 19.	Paramètres du gestionnaire transactionnel	74

ÉQUATIONS DU CHAPITRE 3

Équation 1.	Nombre de possibilités d'exécutions en série de n transactions	39
-------------	--	----

FIGURES DU CHAPITRE 3

Figure 19. Exemple d'écriture d'une transaction.....	31
Figure 20. Exemple d'exécution concurrente en mise à jour immédiate	32
Figure 21. Exemple d'exécution concurrente mise à jour différée	33
Figure 22. Exemple de perte de mise à jour	35
Figure 23. Exemple de lecture impropre (données incohérentes)	36
Figure 24. Exemple de lecture impropre (données non confirmées)	37
Figure 25. Exemple de lecture non-reproductible	37
Figure 26. Réécriture sans problème de l'exemple de perte de mise à jour	38
Figure 27. Exemple d'équivalence : exécution concurrente E_1	40
Figure 28. Exemple d'équivalence : exécution en série E_2 équivalente à E_1	41
Figure 29. Exemple de non-équivalence : exécution en série E_3 non-équivalente à E_2 ni E_1	42
Figure 30. Exemple de graphe de précédence	48
Figure 31. Rappel de l'exemple de graphe de précédence	49
Figure 32. Autre exemple de graphe de précédence.....	50
Figure 33. Graphe de précédence de E_1 ne contenant aucun cycle	51
Figure 34. Graphe de précédence de E_2 contenant un cycle.....	51
Figure 35. Rappel de l'exemple de perte de mise à jour	52
Figure 36. Graphe de précédence de l'exemple de perte de mise à jour	52
Figure 37. Rappel de l'exemple de lecture impropre (données incohérentes).....	52
Figure 38. Graphe de précédence de l'exemple de données incohérentes	53
Figure 39. Rappel de l'exemple de lecture impropre (données non confirmées).....	53
Figure 40. Graphe de précédence de l'exemple de données non confirmées.....	53
Figure 41. Rappel de l'exemple de lecture non-reproductible	53
Figure 42. Graphe de précédence de l'exemple de lecture non reproductible	54
Figure 43. Exemple d'utilisation des événements de verrouillage/déverrouillage	56
Figure 44. Exemple de verrouillage dégradant les performances	57
Figure 45. Exemple de lecture non-reproductible malgré les verrouillages	58
Figure 46. Exemple de transaction à 2 phases.....	59
Figure 47. Exemple de perte de mise à jour réécrit avec des T2P	61
Figure 48. Graphe de précédence de l'exemple de perte de mise à jour (T2P)	61
Figure 49. Exemple de lecture impropre (données incohérentes) réécrit avec des T2P.....	61
Figure 50. Graphe de précédence de l'exemple de lecture impropre (données incohérentes) (T2P)	62
Figure 51. Exemple de lecture impropre (données non confirmées) réécrit avec des T2P.....	62
Figure 52. Graphe de précédence de l'exemple lecture impropre (données non confirmées) (T2P)	62
Figure 53. Exemple de lecture non reproductible réécrit avec des T2P	62
Figure 54. Graphe de précédence de l'exemple lecture non reproductible (T2P).....	63
Figure 55. Exemple d'interblocage	63
Figure 56. Exemple de graphe d'attente montrant un interblocage	64
Figure 57. Graphe d'attente de la perte de mise à jour réécrite en T2P	65
Figure 58. Graphe d'attente de la lecture impropre de données incohérentes réécrite en T2P	65
Figure 59. Graphe d'attente de la lecture impropre de données non confirmées réécrite en T2P	65
Figure 60. Graphe d'attente de la lecture non reproductible réécrite en T2P	65
Figure 61. Exemple de transaction à deux phases amoindrissant les accès concurrentiels.....	66
Figure 62. Exemple d'utilisation des opérations de reclassement.....	68
Figure 63. Niveaux de granularité « théoriques » dans une BD.....	69
Figure 64. Exemple de problème d'objet fantôme	72
Figure 65. Exemple de problème d'objet fantôme résolu grâce au verrouillage intentionnel	73

Usuellement, plusieurs manipulations peuvent être demandées simultanément sur une même BD, notamment de bases de données multiutilisateurs. Ainsi, pour gagner en temps de traitement, un SGBD est généralement capable d'exécuter plusieurs manipulations sur une même base « en même temps »²¹ : on parle alors de **concurrence** ou d'**accès concurrentiels**.

3.1 Notion de transaction

C'est le gestionnaire transactionnel du SGBD qui assure que la simultanéité de ces manipulations se déroule correctement, *i.e.* sans qu'elles viennent interférer les unes avec les autres. Tout repose donc, à la base, sur le concept de transaction...



Définition : « transaction »

Une transaction est un programme ou un fragment de programme composé d'une ou plusieurs instructions mais devant être considéré comme « un tout indissociable » et faisant passer une BD d'un état cohérent à un autre état cohérent (mais il est possible que la base ne soit pas dans un état cohérent pendant l'exécution de la transaction).



Définition : « exécution concurrente » et « accès concurrentiel »

Les SGBD, notamment multiutilisateurs, permettent l'exécution « simultanée » de transactions. On appelle cela une **exécution concurrente** de transactions. Une telle exécution concurrente implique des demandes d'accès « simultanées », par différentes transactions concurrentes, à une même donnée : on parle alors d'**accès concurrentiel** par ces transactions à cette donnée.



Rappel

Une BD est dans un « état cohérent » lorsque toutes les contraintes qui y sont définies sont respectées (on parle bien des contraintes de type ET des contraintes d'intégrité ET des contraintes événementielles).



En pratique

Toutes les transactions gérées par le SGBD partagent tout un ensemble de choses, notamment :

- Une ou des bases de données, bien sûr,
- Les relations et index définis dans ces bases de données,
- La mémoire cache (générale au SGBD).

Elles ont cependant chacune un espace mémoire qui leur est réservé en mémoire de travail et qui leur est donc propre. C'est dans cet espace mémoire qui leur est dédié qu'elles vont stocker des données temporaires et travailler sur leur valeur.

²¹ Comme souvent, il s'agit plutôt de « pseudo-parallélisme ». Les opérations élémentaires ne sont pas vraiment réalisées « en même temps »²¹ mais sont « intercalées » sur le cœur de la machine. Cela permet de donner l'impression que les instructions, composées de ces opérations élémentaires, sont réalisées « simultanément ».

Une transaction est donc une suite d'**opérations**. On « range » ces opérations dans 2 catégories :

- *Des opérations qui n'interagissent pas avec la base de données ni avec le gestionnaire transactionnel* : ces opérations ne sont pas « prises en compte »²² par le gestionnaire de transactions, il s'agit par exemple...
 - D'opérations d'entrées/sorties (saisies/affichages : input varA, display varA, ...),
 - D'opérations de calcul (par exemple varA := varB + varC, ...),
 - ...
- *Des opérations qui interagissent avec la base de données ou avec le gestionnaire transactionnel lui-même* : celles-là sont « prises en compte » pour la gestion des transactions, il s'agit par exemple...
 - D'opérations de début, de confirmation ou d'annulation de transaction (start, commit, rollback),
 - D'opérations de lecture d'une donnée depuis la base de données vers l'espace mémoire dédié à la transaction (read A in varA),



Remarque

Quand on parle de lecture d'une « donnée », on parle de la lecture d'une valeur d'attribut, mais aussi de la lecture d'un n-uplet, d'une table voire de toute la BD.

- D'opérations d'écriture d'une donnée depuis l'espace mémoire dédié à la transaction vers la base de données (write A from varA),



Remarque

De même, quand on parle d'écriture d'une « donnée », on parle de l'écriture d'une valeur d'attribut, mais aussi de l'écriture d'un n-uplet, d'une table voire de toute la BD.

- D'opérations de demande de verrouillage, de déverrouillage, de demande de surclassement ou encore de déclassement d'un verrou (lock X A, unlock A, upgrade A, downgrade A),
- ...

Cette deuxième catégorie d'opérations constitue ce que l'on appelle les **événements transactionnels**.



Définition : « événement transactionnel » ou « opération événementielle » ou « événement »

Un événement transactionnel est une opération faisant partie d'une transaction interagissant avec la base de données ou avec le gestionnaire transactionnel.

²² En fait, si, elles sont bien sûr « prises en compte » puisqu'elles vont être exécutées. Par contre, leur présence n'a pas d'impact sur la gestion de l'accès concurrentiel aux données.

Nous considérerons tout d'abord les 5 événements transactionnels suivants :

Événement	Signification
start	Démarrage de la transaction courante
read D in varD	Lecture d'une donnée D depuis la BD (sa valeur est stockée dans l'espace mémoire dédié à la transaction courante, dans la variable varD)
write D from varD	Modification d'une donnée D dans la BD (la valeur écrite est celle de la variable varD qui se trouve dans l'espace mémoire dédié à la transaction courante)
rollback	Annulation de la transaction courante
commit	Confirmation de la transaction courante

Tableau 2. Événements transactionnels basiques

Exemple

Reprendons le fameux exemple du transfert d'une somme S d'un compte A vers un compte B. Avec ces 5 événements transactionnels, l'écriture de la transaction T réalisant ce transfert pourrait être la suivante (les événements transactionnels sont écrits en rouge) :

T	
1	start
2	read S in varS
3	read A in varA
4	varA := varA - varS
5	write A from varA
6	read B in varB
7	varB := varB + varS
8	write B from varB
9	commit

Figure 16. Exemple d'écriture d'une transaction

L'exemple précédent montre bien une des problématiques (probablement la plus importante) posée par l'accès concurrentiel aux données : on visualise aisément que si une transaction concurrente vient modifier en base les données A, B ou S pendant que la transaction T ci-dessus s'exécute, on aura finalement des données incohérentes au sein de la base de données !

Le gestionnaire transactionnel va, bien sûr et heureusement, pouvoir mettre en place des mécanismes dédiés à la résolution (voire à l'évitement) de ce genre de problèmes (ainsi que d'autres). Mais, avant d'aborder tout cela, il faut dans un premier temps considérer la façon dont les modifications sont faites par une transaction sur des données d'une base de données sont effectivement propagées au sein de la base de données...

3.2 Propagation des modifications

Lorsqu'une opération de modification de donnée est exécutée dans une transaction T , c'est le gestionnaire transactionnel qui se charge de la « propagation » de cette modification.



Définition : « mode de mise à jour »

Le **mode de mise à jour** est la technique de propagation des modifications mise en œuvre par un gestionnaire transactionnel.

Il existe 2 façons de réaliser cette propagation, donc 2 modes de mise à jour...

3.2.1 Mode de mise à jour immédiate

Lorsqu'un gestionnaire transactionnel est en mode de mise à jour immédiate, toute demande de modification d'une donnée D par une transaction T est immédiatement propagée dans la base de données : le gestionnaire transactionnel s'appuie alors sur le gestionnaire de mémoire cache pour écrire dans la BD la nouvelle valeur de la donnée D . L'impact de ce mode de mise à jour est triple :

- Toute modification est immédiatement visible par toutes les transactions concurrentes (sans parler des transactions qui démarreront après la modification),
- Tant que la transaction T n'est pas terminée (par un `commit` ou un `rollback`), toute modification qu'elle a demandée est tout de même propagée dans la base,
- Si la transaction T vient à être annulée (par un `rollback`), le gestionnaire de transactions doit annuler les modifications qu'elle a demandées avant cette annulation : pour ce faire, toujours en s'appuyant sur le gestionnaire de mémoire cache, pour toute donnée modifiée par T , il réécrit immédiatement dans la BD la valeur de cette donnée avant que T ne la modifie.



Exemple

Soit l'exécution concurrente E de 2 transactions T_1 et T_2 ci-dessous, en supposant que l'on est en mode de mise à jour immédiate...

E			Remarques
	T_1	T_2	
1	start		$A = 10, B = 50$
2	read A in varA ₁		T_1 lit $A = 10$
3	read B in varB ₁		T_1 lit $B = 50$
4	varC ₁ := varA ₁ * 2		
5		start	5
6		read A in varA ₂	T_2 lit $A = 10$
7	write B from varC ₁		T_1 écrit $B = 20$
8		read B in varB ₂	T_2 lit $B = 20$
9

Figure 17. Exemple d'exécution concurrente en mise à jour immédiate

3.2.2 Mode de mise à jour différée

Lorsqu'un gestionnaire transactionnel est en mode de mise à jour différée, une demande de modification d'une donnée D par une transaction T n'est pas immédiatement propagée dans la base de données : elle n'est dans un premier temps visible que par cette transaction T (la modification n'est en fait réalisée que dans l'espace mémoire dédié à cette transaction). Ce n'est que lorsque cette transaction est confirmée²³ (par un `commit`) que les modifications qu'elle a faites sont propagées dans la BD : pour ce faire, là encore, le gestionnaire transactionnel s'appuie sur le gestionnaire de mémoire cache pour réaliser ces propagations. L'impact de ce mode de mise à jour est également triple :

- Toute modification faite par une transaction T reste invisible à toute transaction concurrente (ou qui démarra après la demande de modification) jusqu'à ce que la transaction T soit confirmée (par un `commit`),
- Ce n'est que quand la transaction T est confirmée (par un `commit`) que ses modifications sont propagées, toutes « d'un bloc », dans la base,
- Si la transaction T vient à être annulée (par un `rollback`), il n'y a aucune modification à annuler dans la BD (puisque aucune modification réalisée par T n'y a été propagée).

Exemple

Soit l'exécution concurrente E de 2 transactions T_1 et T_2 ci-dessous, en supposant que l'on est en mode de mise à jour différée...

E		Remarques
	T ₁	T ₂
1	start	
2	read A in varA ₁	
3	read B in varB ₁	
4	varC ₁ := varA ₁ * 2	
5		start
6		read A in varA ₂
7	write B from varC ₁	
8		read B in varB ₂
9	varC ₁ := varB ₁ / 2	
10	write C from varC ₁	
11	commit	
12		read C in varC ₂
13		...

Figure 18. Exemple d'exécution concurrente mise à jour différée



Remarque

Sauf indication contraire explicite, nous supposerons dans la suite de ce chapitre que nous sommes en mode de mise à jour immédiate. **Tout ce qui sera montré pourrait également l'être en mode de mise à jour différée** (mais ce serait souvent plus long et/ou plus complexe).



²³ Si elle l'est « un jour », ça n'est pas obligatoire (puisque elle peut aussi être annulée par un `rollback`).

3.3 Problèmes dus à la concurrence

Nous en avons déjà parlé, un SGBD doit assurer que toute transaction respecte les propriétés dites « ACID »... Ce respect doit bien sûr être assuré quel que soit le mode de mise à jour adopté par le gestionnaire transactionnel !



Rappel

Ces propriétés sont les suivantes :

- **Atomicité** : soit toutes les modifications effectuées par une transaction sont enregistrées dans la BD, soit aucune ne l'est. Ainsi :
 - Si une transaction est confirmée (`commit`), toutes les modifications qu'elle a effectuées sont enregistrées dans la BD et rendues visibles aux autres utilisateurs,
 - Si une transaction est interrompue (`rollback` ou panne), alors aucune de ces modifications n'est enregistrée dans la BD (les modifications déjà effectuées doivent être annulées).
- **Cohérence** : une transaction fait passer une BD d'un état cohérent à un autre état cohérent.
- **Isolation** : une transaction se déroule sans être perturbée par les transactions concurrentes : tout se passe comme si elle se déroulait seule.
- **Durabilité** : une fois qu'une transaction a été confirmée (`commit`), le SGBD garantit qu'aucune modification qu'elle a effectuée ne sera perdue, quels que soient les accidents qui surviendront : interruption, panne du système d'exploitation, « crash » disque, ...

Nous l'avons également déjà vu, plusieurs transactions peuvent se dérouler en même temps : on dit qu'elles sont **concurrentes**. Si aucun contrôle du déroulement des transactions n'est mis en place par le gestionnaire transactionnel, 4 types de problèmes peuvent se rencontrer lors de l'exécution de transactions en concurrence :

- Perte de mise à jour,
- Lecture impropre :
 - Lecture de données incohérentes,
 - Lecture de données non confirmées,
- Lecture non reproductible.

Nous allons illustrer ces 4 types de problèmes au travers d'exemples²⁴.



En pratique

Une des propriétés ACID à faire respecter est la propriété d'**isolation** : bien que les transactions s'exécutent en concurrence, elles doivent s'exécuter « comme si elles étaient seules au monde » prises indépendamment les unes des autres. En pratique, les 4 types de problèmes illustrés ci-après sont dus à un non-respect de la propriété d'**isolation** (pouvant éventuellement, ou non, entraîner un non-respect d'autres propriétés ACID, notamment la cohérence).

²⁴ Ces exemples étant réalisés en supposant que l'on est en mode de mise à jour immédiate (mais, sauf exception, on pourrait aussi illustrer ces types de problèmes avec des exemples réalisés en supposant que l'on est en mode de mise à jour différée).

3.3.1 Perte de mise à jour

Le problème de la perte de mise à jour correspond en fait à une modification faite par une transaction et « écrasée » par une transaction concurrente (comme si la première modification n'avait jamais eu lieu).



Exemple

Soit l'exécution concurrente E de 2 transactions T_1 et T_2 ci-dessous, en supposant que l'on est en mode de mise à jour immédiate...

E		Remarques
	T ₁	T ₂
1	start	1
2	read A in varA ₁	2 T_1 lit A = 10
3		start
4		read A in varA ₂
5	varA ₁ := varA ₁ + 10	5
6	write A from varA ₁	6 T_1 écrit A = 20
7		varA ₂ := varA ₂ + 50
8		write A from varA ₂ T₂ écrit A = 60
9

Figure 19. Exemple de perte de mise à jour

Après l'écriture de A par T_2 on devrait avoir A = 70 puisque T_1 a ajouté 10 à A ET T_2 lui a ajouté 50. Mais l'ordre des opérations est tel que A = 60 : la mise à jour demandée par T_1 est « écrasée »... Dans cette exécution concurrente, le problème est principalement dû au non-respect de la propriété d'isolation : si T_1 et T_2 s'étaient réellement exécutées indépendamment l'une de l'autre, le problème n'aurait pas été rencontré.

3.3.2 Lecture impropre (données incohérentes)

Une des propriétés ACID, que le gestionnaire transactionnel doit faire respecter, est la cohérence : cette propriété indique qu'une transaction fait passer la BD d'un état cohérent à un autre état cohérent.



Remarque

Bien sûr, l'état de la BD peut être incohérent pendant l'exécution d'une transaction, mais uniquement si c'est cette transaction qui a modifié des données présentant une incohérence : si ce n'est pas elle qui les a modifiées, c'est comme si elle avait démarré alors que la BD n'était pas dans un état cohérent.

À cause de l'accès concurrentiel, et donc de la possibilité d'exécuter des transactions de façon concurrente, il peut fréquemment arriver que cette propriété ne soit pas respectée...

Exemple

Soit l'exécution concurrente E de 2 transactions T_1 et T_2 ci-dessous, en supposant que l'on est en mode de mise à jour immédiate... On a une contrainte événementielle qui vérifie (pour pas mal d'opérations²⁵) que $A + B = 200$.

E		Remarques
	T ₁	T ₂
1	start	1
2	read A in varA ₁	2
3	varA ₁ := varA ₁ - 50	3
4	write A from varA ₁	4
5		start 5
6		read A in varA ₂ 6
7		read B in varB ₂ 7
8		varC ₂ := varA ₂ + varB ₂ 8
9		display varC ₂ 9 Affiche 150 !
10	read B in varB ₁	10 T ₁ lit B = 80
11	varB ₁ := varB ₁ + 50	11
12	write B from varB ₁	12 T ₁ écrit B = 130
13	...	13

Figure 20. Exemple de lecture impropre (données incohérentes)

La transaction T_1 est une transaction cohérente vis-à-vis de la contrainte d'intégrité puisqu'elle retranche à A ce qu'elle ajoute à B . En revanche, la transaction T_2 ne modifiant ni A ni B , elle devrait afficher la valeur 200. Mais l'ordre des opérations est tel que T_2 affiche 150. La cohérence n'est ici pas respectée parce que l'isolation ne l'est pas elle-même...

3.3.3 Lecture impropre (données non confirmées)

La lecture de données non confirmées est due à une transaction qui travaille sur des valeurs lues alors que ces valeurs ont précédemment été modifiées par une transaction concurrente et que ces modifications vont être « défaites » puisque ladite transaction concurrente va être annulée (par un rollback).



Remarque

Ce problème ne peut PAS être rencontré si on est en mode de mise à jour différée !

²⁵ Cette contrainte n'est par exemple pas vérifiable entre une modification de A et la modification de B correspondante, c'est évident...

Exemple

Soit l'exécution concurrente E de 2 transactions T_1 et T_2 ci-dessous, en supposant que l'on est en mode de mise à jour immédiate...

E		Remarques
	T_1	T_2
1		start
2		$\text{varA}_2 := 70$
3		write A from varA_2
4	start	
5	read A in varA_1	
6		rollback
7	...	

Figure 21. Exemple de lecture impropre (données non confirmées)

La transaction T_1 va continuer à travailler par rapport à une valeur de A qui n'est finalement pas la bonne (puisque la valeur de A est restaurée à une précédente valeur lors de l'annulation de la transaction T_2) : la transaction T_1 a lu une valeur de A « incorrecte » car tout aurait dû se passer comme si T_2 n'avait jamais changé A. C'est donc comme si l'annulation de la transaction T_2 n'était pas prise en compte par la transaction T_1 alors que, pourtant, cette annulation a des effets sur la BD et, donc, sur des données pouvant être manipulées par la transaction T_1 (comme ici la donnée A). Là encore, on peut observer que la propriété d'isolation n'est pas respectée...

3.3.4 Lecture non reproductible

La lecture non-reproductible survient lorsqu'une transaction lit plusieurs fois une même donnée²⁶ sans elle-même l'avoir modifiée entretemps. Dans ce cas, le principe d'isolation indique que chaque lecture de la même donnée doit lire la même valeur pour cette donnée, tant que l'on ne l'a pas soi-même modifiée. Malheureusement, en raison de l'accès concurrentiel, ça n'est pas toujours le cas...

Exemple (début)

Soit l'exécution concurrente E de 2 transactions T_1 et T_2 ci-dessous, en supposant que l'on est en mode de mise à jour immédiate...

E		Remarques
	T_1	T_2
1		start
2		read A in varA_2
3	start	
4	$\text{varA}_1 := 20$	
5	write A from varA_1	
6		read A in varA_2
7	...	

Figure 22. Exemple de lecture non-reproductible

²⁶ Peu importe la raison.



Exemple (fin)

La transaction T_2 lit 2 fois la donnée A (peu importe la raison) sans elle-même avoir modifié cette donnée entre les 2 lectures. Pourtant, lors de la seconde lecture, elle ne lit pas la même valeur que lors de la première lecture. Encore une fois, cela est un effet du non-respect de la propriété d'isolation : la modification de la donnée A faite par la transaction T_1 entre les 2 lectures faites par la transaction T_2 vient perturber la seconde lecture de cette donnée par la transaction T_2 .

3.3.5 Comment éviter ces problèmes d'accès concurrentiels ?

Nous l'avons vu dans chacun de ces 4 types de problèmes, ceux-ci sont notamment dus au non-respect de la propriété d'isolation. En effet, au lieu de s'exécuter en parallèle mais « comme si elles étaient seules au monde », précisément comme l'exige la propriété d'isolation, les transactions s'exécutent en parallèle mais en interférant les unes avec les autres et ce sont bien ces interférences qui sont la source des soucis rencontrés. En revanche, aucun problème n'aurait été rencontré si ces transactions avaient été exécutées les unes à la suite des autres !



Exemple

Dans le cas de l'exemple de la perte de mise à jour (cf. §3.3.1), on aurait par exemple eu l'exécution concurrente suivante dans laquelle la transaction T_1 est complètement exécutée avant que la transaction T_2 ne démarre :

	E		Remarques
	T_1	T_2	
1	start		$A = 10$
2	read A in varA ₁		T_1 lit A = 10
3	varA ₁ := varA ₁ + 10		
4	write A from varA ₁		T_1 écrit A = 20
5	...		
6	commit		
7		start	
8		read A in varA ₂	T_2 lit A = 20
9		varA ₂ := varA ₂ + 50	
10		write A from varA ₂	T_2 écrit A = 70
11	

Figure 23. Réécriture sans problème de l'exemple de perte de mise à jour

Cette fois, la modification de la donnée A réalisée par la transaction T_1 est bien sûr prise en compte durant l'exécution de la transaction T_2 .

De même, les problèmes montrés dans les autres exemples (lecture impropre de données incohérentes, lecture impropre de données non confirmées et lecture non reproductible, cf. respectivement §3.3.2, §3.3.3 et §3.3.4) ne se poseraient plus si l'une des 2 transactions des exécutions concurrentes présentées était complètement exécutée avant l'autre.

Une telle exécution où les transactions sont en fait réalisées les unes à la suite des autres est appelée une **exécution en série**. Du coup, on peut aussi définir formellement le concept d'exécution concurrente de transactions !



Définition : « exécution de transactions en série »

Une exécution de transactions est appelée **exécution en série** si, pour chaque transaction T qui la compose, l'ensemble des opérations de T est réalisé sans être « interrompu » par l'exécution d'une ou plusieurs opérations d'une autre transaction. Autrement dit, l'exécution d'un ensemble de transactions est dite **en série** si, pour tout couple de transactions de cette exécution, tous les événements de l'une précédent tous les événements de l'autre.



Remarque

Notez qu'il est possible d'exécuter en série un même ensemble de transactions de plusieurs façons ! Ainsi :

- 2 transactions T_1 et T_2 peuvent être exécutées en série de 2 façons :
 - L'exécution en série E_1 où T_1 commence et T_2 termine,
 - L'exécution en série E_2 où T_2 commence et T_1 termine.
- 3 transactions T_1 , T_2 et T_3 peuvent être exécutées en série de 6 façons :
 - L'exécution en série E_1 correspondant à T_1 puis T_2 puis T_3 ,
 - L'exécution en série E_2 correspondant à T_1 puis T_3 puis T_2 ,
 - L'exécution en série E_3 correspondant à T_2 puis T_1 puis T_3 ,
 - L'exécution en série E_4 correspondant à T_2 puis T_3 puis T_1 ,
 - L'exécution en série E_5 correspondant à T_3 puis T_1 puis T_2 ,
 - L'exécution en série E_6 correspondant à T_3 puis T_2 puis T_1 .
- ...
- n transactions T_1, \dots, T_n peuvent être exécutées en série de Nb_{ES} façons où :

$$Nb_{ES}(T_1, \dots, T_n) = n!$$

Équation 1. Nombre de possibilités d'exécutions en série de n transactions



Définition : « exécution concurrente de transactions »

Une exécution de transactions est appelée **exécution concurrente** si au moins 1 opération d'une transaction T de cette exécution est effectuée entre la réalisation d'opérations appartenant à une (d'autres) transaction(s) de cette même exécution.

On peut donc dire qu'**une exécution en série d'un ensemble de transactions respecte forcément la propriété d'isolation** et, donc, ne cause aucun des problèmes d'accès concurrentiel aux données identifiés précédemment. D'où la question suivante...



Question

Puisque cela règle les problèmes énoncés plus haut, pourquoi, alors, ne pas exécuter les transactions complètement les unes à la suite des autres, *i.e.* en série ?

Réponse

Tout simplement parce qu'on perdrait alors le bénéfice, en termes de performances notamment, de leur exécution en parallèle, *i.e.* en concurrence. Ce bénéfice étant loin d'être négligeable, il est exclu de s'en passer complètement !

Pour autant, la situation n'est pas insoluble : l'idée va être de mettre en œuvre un mécanisme, appelé le **contrôle de concurrence**, qui a pour objectif d'essayer de maintenir de bonnes performances en traitant des exécutions concurrentes tout en essayant tout de même de « se rapprocher » d'une exécution en série des mêmes transactions. Cet objectif peut paraître, avec ce que l'on a dit plus haut, totalement contre-intuitif mais il ne l'est en fait pas tant que ça...

3.4 Contrôle de concurrence

Le contrôle de concurrence est un des services rendus au SGBD par le gestionnaire transactionnel. C'est donc ce dernier qui se charge de l'ensemble des points abordés ici (vu que c'est également lui qui se charge du traitement des transactions, cela est très logique). Ce contrôle repose principalement sur 2 notions : la sérialisabilité d'une exécution concurrente de transactions (cf. §3.4.1) et le verrouillage des données (cf. §3.4.2).

3.4.1 Sérialisabilité d'une exécution concurrente de transactions

L'idée est de mettre en œuvre une exécution concurrente de transactions tout en vérifiant que l'exécution mise en œuvre est « équivalente » à une exécution en série de ces mêmes transactions. Reste à savoir ce que signifie que 2 exécutions de transactions sont « équivalentes »...

3.4.1.1 Équivalence d'exécutions de transactions

Cette équivalence est formellement définie comme suit...



Définition : « équivalence d'exécutions de transactions »

Deux exécutions de transactions E_1 et E_2 sont **équivalentes** si et seulement si :

- Elles sont constituées des mêmes opérations, dans le même ordre au sein de chacune des transactions (*i.e.* quand on regarde les opérations de chacune des transactions indépendamment des autres transactions),
- Elles produisent le même état final de la BD et les mêmes résultats pour les transactions (*i.e.* les lectures retournent les mêmes « valeurs » et les écritures sur une même donnée sont réalisées dans le même ordre).



Exemple (début)

Soit l'exécution concurrente E_1 ci-dessous :

E ₁		Remarques
	T ₁	T ₂
1	start	1
2	read A in varA ₁	2
3	varA ₁ := varA ₁ + 10	3
4	write A from varA ₁	4
5		start
6		read A in varA ₂
7	read B in varB ₁	6
8		7
9		T ₂ lit A = 10
10	varA ₂ := varA ₂ + 50	8
11	varB ₁ := varB ₁ + 10	9
12	write A from varA ₂	T ₂ écrit A = 60
13	write B from varB ₁	10
14		11
15	read B in varB ₂	T ₁ écrit B = 10
16	varB ₂ := varB ₂ + 50	12
		T ₂ lit B = 10
		13
		14
		T ₂ écrit B = 60
	commit	15
		16
	commit	

Figure 24. Exemple d'équivalence : exécution concurrente E₁

Exemple (fin)

L'exécution concurrente E_1 ci-dessus est équivalente à l'exécution en série E_2 ci-dessous :

- Dans les 2 exécutions E_1 et E_2 , les opérations de chaque transaction (prises indépendamment l'une de l'autre) sont les mêmes et dans le même ordre,
- Les lectures faites dans E_2 retournent les mêmes valeurs que les lectures faites dans E_1 ,
- Les écritures faites dans E_2 sont réalisées dans le même ordre que dans E_1 , ce pour chaque donnée A et B,
- L'état final de la BD est le même dans les 2 cas ($A = B = 60$).

E_2		Remarques
T_1	T_2	
1 start		1
2 read A in varA ₁		2 T_1 lit A = 0
3 varA ₁ := varA ₁ + 10		3
4 write A from varA ₁		4 T_1 écrit A = 10
5 read B in varB ₁		5 T_1 lit B = 0
6 varB ₁ := varB ₁ + 10		6
7 write B from varB ₁		7 T_1 écrit B = 10
8 commit		8
9 start		9
10 read A in varA ₂	10	T_2 lit A = 10
11 varA ₂ := varA ₂ + 50	11	
12 write A from varA ₂	12	T_2 écrit A = 60
13 read B in varB ₂	13	T_2 lit B = 10
14 varB ₂ := varB ₂ + 50	14	
15 write B from varB ₂	15	T_2 écrit B = 60
16 commit	16	

Figure 25. Exemple d'équivalence : exécution en série E_2 équivalente à E_1

Remarque (début)

Les différentes exécutions en série possibles d'un même ensemble de transactions ne sont pas forcément toutes équivalentes entre elles ! Soit, par exemple, l'exécution en série E_3 des mêmes transactions que dans l'exemple ci-dessus. Cette exécution E_3 n'est en effet pas équivalente à l'exécution en série E_2 des mêmes transactions. Elle n'est pas équivalente non plus, du coup, à l'exécution concurrente E_1 de ces mêmes transactions. En effet, on constate :

- Dans les exécutions E_1 , E_2 et E_3 , les opérations de chaque transaction (prises indépendamment l'une de l'autre) sont les mêmes et dans le même ordre,
- Les lectures faites dans E_3 ne retournent pas les mêmes valeurs que les lectures faites dans E_1 ni dans E_2 ,
- Pour chaque donnée, les écritures faites dans E_3 ne sont pas réalisées dans le même ordre que dans E_1 ni dans E_2 ,
- L'état final de la BD est le même dans les 2 cas ($A = B = 60$).

Certaines des propriétés nécessaires pour que les exécutions soient équivalentes ne sont pas respectées : elles ne sont donc pas équivalentes.



Remarque (fin)

L'exécution en série E_3 est la suivante :

E_3		Remarques
	T_1	T_2
1		start
2		read A in varA ₂
3		varA ₂ := varA ₂ + 50
4		write A from varA ₂
5		read B in varB ₂
6		varB ₂ := varB ₂ + 50
7		write B from varB ₂
8		commit
9	start	
10	read A in varA ₁	
11	varA ₁ := varA ₁ + 10	
12	write A from varA ₁	
13	read B in varB ₁	
14	varB ₁ := varB ₁ + 10	
15	write B from varB ₁	
16	commit	

Figure 26. Exemple de non-équivalence : exécution en série E_3 non-équivalente à E_2 ni E_1

3.4.1.2 Sérialisabilité d'une exécution concurrente de transactions

Il reste donc à savoir comment savoir si une exécution concurrente de transactions (puisque l'on souhaite optimiser les performances, on reste bien sur des exécutions concurrentes) est équivalente à une exécution en série de ces mêmes transactions.



Définition : « sérialisabilité d'une exécution concurrente de transactions »

La sérialisabilité d'une exécution concurrente de transactions est sa capacité à être équivalente à une exécution en série du même ensemble de transactions.



Définition : « exécution sérialisable »

Une exécution concurrente de transactions est sérialisable si et seulement s'il existe au moins une exécution en série du même ensemble de transactions qui lui est équivalente.

L'intérêt est le suivant : **il est démontré qu'une exécution concurrente de transaction sérialisable ne pose aucun des problèmes d'accès concurrentiels énoncés plus haut**. C'est logique : étant donné qu'elle est équivalente à une exécution en série des mêmes transactions et qu'une exécution en série respecte la propriété d'isolation, les soucis énoncés ne peuvent pas se produire.

Il est donc nécessaire de pouvoir déterminer simplement si une exécution concurrente est sérialisable (ou non). Si c'est le cas, aucun problème d'accès concurrentiel ne se posera !



En pratique

Pour montrer qu'une exécution concurrente est sérialisable, il suffit de montrer au moins une des exécutions en série du même ensemble de transactions qui lui est équivalente. En revanche, pour montrer qu'une exécution concurrente n'est PAS sérialisable, il faut montrer qu'elle n'est équivalente à AUCUNE des exécutions en série possibles du même ensemble de transactions.

3.4.1.3 Conditions de sérialisabilité d'une exécution concurrente

Déterminer si une exécution concurrente E_c est équivalente à une exécution en série E_s du même ensemble de transactions revient à savoir s'il existe une suite de **permutations** (ou **commutations**) des opérations constituant E_c permettant d'arriver à E_s . Ces permutations entre opérations doivent suivre certaines contraintes.



Remarque

Les opérations transactionnelles que l'on considère jusque-là sont :

- Le début d'une transaction (`start`), qui ne provoque ni lecture ni écriture,
- La lecture d'une donnée (`read A in varA`) par une transaction,
- L'écriture d'une donnée (`write A from varA`) par une transaction :
 - En mode de mise à jour immédiate, elle provoque une écriture,
 - En mode de mise à jour différée, elle ne provoque AUCUNE écriture !
- La confirmation d'une transaction (`commit`) :
 - En mode de mise à jour immédiate, elle ne provoque ni lecture ni écriture,
 - En mode de mise à jour différée, elle peut provoquer des écritures (si des `write` figurent dans la même transaction),
- L'annulation d'une transaction (`rollback`) :
 - En mode de mise à jour immédiate, elle peut provoquer des écritures (si des `write` figurent dans la même transaction),
 - En mode de mise à jour différée, elle ne provoque ni lecture ni écriture.

On va donc simplifier en considérant ici :

- Des opérations sans accès à la BD op_{NoBD} , sans lecture ni écriture,
- Des opérations de lecture depuis la BD op_{Lect} ,
- Des opérations d'écriture vers la BD op_{Ecrit} .

Mode de mise à jour	Opérations transactionnelles				
	start	read	write	commit	rollback
Immédiate	op_{NoBD}	op_{Lect}	op_{Ecrit}	op_{NoBD}	op_{Ecrit} ou op_{NoBD}
Différée	op_{NoBD}	op_{Lect}	op_{NoBD}	op_{Ecrit} ou op_{NoBD}	op_{NoBD}

Tableau 3. Nature des opérations transactionnelles selon le mode de mise à jour



Définition : « opérations permutable »

Deux opérations d'une exécution E sont permutable si et seulement si :

- Ces 2 opérations appartiennent à 2 transactions T_1 et T_2 différentes (on ne peut donc pas permuter 2 opérations d'une même transaction²⁷),
- Si on ne considère que les opérations faisant partie des 2 transactions T_1 et T_2 (donc en « ignorant » les opérations de toutes les autres transactions), ces 2 opérations se suivent immédiatement (donc aucune autre opération de T_1 ni de T_2 ne doit se trouver entre elles),
- Ces 2 opérations sont non conflictuelles.

Reste donc à savoir si 2 opérations sont conflictuelles ou non...



Définition : « opérations non-conflictuelles »

Deux opérations d'une exécution E sont non-conflictuelles si et seulement si leur échange n'entraîne²⁸ :

- Aucune modification des valeurs retournées par les éventuelles lectures,
- Pour chaque donnée manipulée, les écritures sont faites dans le même ordre,
- Aucun changement dans l'état final des données.

On va donc considérer les 9 cas possibles de permutations (chaque cas peut se lire dans les 2 sens) :

1. Permutation de 2 opérations op_{NoBD} n'accédant pas à la BD : les 2 opérations n'accédant pas à la BD, les 3 propriétés de non-conflictualité sont ici respectées.

Avant permutation		Après permutation	
T_1	T_2	T_1	T_2
op_{NoBD}			op_{NoBD}
	op_{NoBD}	op_{NoBD}	

Tableau 4. Permutation de 2 opérations op_{NoBD} sans accès BD

2. Permutation d'une opération op_{NoBD} sans accès à la BD et d'une opération de lecture op_{Lect} : une seule des 2 opérations accédant à la BD, les 3 propriétés de non-conflictualité sont également respectées ici.

Avant permutation		Après permutation	
T_1	T_2	T_1	T_2
op_{NoBD}			op_{Lect}
	op_{Lect}	op_{NoBD}	

Tableau 5. Permutation d'une opération op_{NoBD} sans accès BD et d'une lecture op_{Lect}

²⁷ Et c'est heureux ! 😊

²⁸ On retrouve ici des propriétés liées à l'équivalence entre exécutions de transactions, ce qui est fort logique...

3. Permutation d'une opération op_{NoBD} sans accès à la BD et d'une opération d'écriture op_{Ecrit} : là encore, une seule des 2 opérations accédant à la BD, les 3 propriétés de non-conflictualité sont respectées.

Avant permutation		Après permutation	
T ₁	T ₂	T ₁	T ₂
op_{NoBD}			op_{Ecrit}
	op_{Ecrit}	op_{NoBD}	

Tableau 6. Permutation d'une opération op_{NoBD} sans accès BD et d'une écriture op_{Ecrit}

4. Permutation de 2 opérations op_{Lect} de lecture de données différentes A et B : ces 2 opérations accèdent à la BD mais pas à « la même partie » de la BD ; elles ne peuvent donc interférer l'une avec l'autre et les 3 propriétés de non-conflictualité sont respectées.

Avant permutation		Après permutation	
T ₁	T ₂	T ₁	T ₂
$op_{Lect}(A)$			$op_{Lect}(B)$
	$op_{Lect}(B)$	$op_{Lect}(A)$	

Tableau 7. Permutation de 2 lectures op_{Lect} de 2 données différentes

5. Permutation de 2 opérations op_{Lect} de lecture d'une même donnée A : ces 2 opérations accèdent à « la même partie » de la BD mais ne la modifient pas ; elles ne peuvent donc interférer l'une avec l'autre et les 3 propriétés de non-conflictualité sont respectées.

Avant permutation		Après permutation	
T ₁	T ₂	T ₁	T ₂
$op_{Lect}(A)$			$op_{Lect}(A)$
	$op_{Lect}(A)$	$op_{Lect}(A)$	

Tableau 8. Permutation de 2 lectures op_{Lect} d'une même donnée

6. Permutation d'une opération de lecture op_{Lect} d'une donnée A et d'une opération d'écriture op_{Ecrit} d'une donnée B : ces 2 opérations n'accèdent pas à « la même partie » de la BD ; elles ne peuvent donc interférer l'une avec l'autre et les 3 propriétés de non-conflictualité sont respectées.

Avant permutation		Après permutation	
T ₁	T ₂	T ₁	T ₂
$op_{Lect}(A)$			$op_{Ecrit}(B)$
	$op_{Ecrit}(B)$	$op_{Lect}(A)$	

Tableau 9. Permutation d'une lecture op_{Lect} et d'une écriture op_{Ecrit} de 2 données différentes

7. Permutation d'une opération de lecture op_{Lect} et d'une opération d'écriture op_{Ecrit} de la même donnée A : les 2 opérations accèdent à « la même partie » de la BD ET l'une des 2 opérations modifie cette « partie » ; **elles interfèrent donc l'une avec l'autre puisque la lecture ne retourne pas la même valeur dans les 2 cas !** Dans un cas, la lecture retourne la valeur AVANT sa modification alors que, dans l'autre cas, la lecture retourne la valeur APRES sa modification.

Avant permutation		Après permutation	
T ₁	T ₂	T ₁	T ₂
$op_{Lect}(A)$			$op_{Ecrit}(A)$
	$op_{Ecrit}(A)$	$op_{Lect}(A)$	

Tableau 10. Permutation d'une lecture op_{Lect} et d'une écriture op_{Ecrit} d'une même donnée

8. Permutation de 2 opérations op_{Ecrit} d'écriture de données différentes A et B : ces 2 opérations accèdent à la BD mais pas à « la même partie » de la BD ; elles ne peuvent donc interférer l'une avec l'autre et les 3 propriétés de non-conflictualité sont respectées.

Avant permutation		Après permutation	
T ₁	T ₂	T ₁	T ₂
	$op_{Ecrit}(A)$		$op_{Ecrit}(B)$
	$op_{Ecrit}(B)$	$op_{Ecrit}(A)$	

Tableau 11. Permutation de 2 écritures op_{Ecrit} de 2 données différentes

9. Permutation de 2 opérations op_{Ecrit} d'écriture d'une même donnée A : ces 2 opérations accèdent à « la même partie » de la BD ET la modifient ; **elles interfèrent donc l'une avec l'autre puisque l'état final n'est pas le même dans les 2 cas, les 2 écritures de la donnée A ne s'effectuant pas dans le même ordre !** C'est la dernière écriture faite qui est prise en compte (l'autre écriture étant « ignorée »).

Avant permutation		Après permutation	
T ₁	T ₂	T ₁	T ₂
	$op_{Ecrit}(A)$		$op_{Ecrit}(A)$
	$op_{Ecrit}(A)$	$op_{Ecrit}(A)$	

Tableau 12. Permutation de 2 écritures op_{Ecrit} d'une même donnée

Ainsi, pour synthétiser, on a donc...

Conflictualité entre op_1 et op_2		op_2					
		op_{NoBD}	op_{Lect}		op_{Ecrit}		
			A	B	A	B	
op_1	op_{NoBD}		Non	Non	Non	Non	Non
	op_{Lect}	A	Non	Non	Non	Oui	Non
		B	Non	Non	Non	Non	Oui
	op_{Ecrit}	A	Non	Oui	Non	Oui	Non
		B	Non	Non	Oui	Non	Oui

Tableau 13. Synthèse des conflictualités entre types d'opérations transactionnelles

En résumé, **2 opérations transactionnelles sont conflictuelles si elles opèrent sur la même donnée et si au moins l'une des 2 modifie cette donnée.** Elles sont non-conflictuelles sinon.

Ainsi, une exécution concurrente est sérialisable s'il existe une suite de permutations d'opérations non-conflictuelles qui permettrait de la transformer en une exécution en série (implicitement équivalente, donc). Cette transformation est appelée **sérialisation** (de l'exécution concurrente en une exécution en série).



Définition : « sérialisation »

La **sérialisation** d'une exécution concurrente de transactions est réalisée en transformant cette exécution concurrente en une exécution en série par une suite de permutations d'opérations non-conflictuelles. Par définition (des permutations et, surtout, de la non-conflictualité), l'exécution en série issue de la transformation est forcément équivalente à l'exécution concurrente transformée.



Exemple

On l'a montré que l'exécution concurrente E_1 montrée plus haut (cf. Figure 24) est équivalente à l'exécution en série E_2 également montrée plus haut (cf. Figure 25). On aurait pu également le montrer en présentant une suite de permutations d'opérations non-conflictuelles permettant de transformer l'exécution concurrente E_1 afin d'obtenir l'exécution en série E_2 . Par exemple :

- Permutation de `read B de T1` avec `read A de T2`,
- Permutation de `read B de T1` avec `start de T2`,
- Permutation de `write B de T1` avec `write A de T2`,
- Permutation de `write B de T1` avec `read A de T2`,
- Permutation de `write B de T1` avec `start de T2`,
- Permutation de `commit de T1` avec `write B de T2`,
- Permutation de `commit de T1` avec `read B de T2`,
- Permutation de `commit de T1` avec `write A de T2`,
- Permutation de `commit de T1` avec `read A de T2`,
- Permutation de `commit de T1` avec `start de T2`.

En revanche, on ne pourra jamais avoir l'exécution en série E_3 (cf. Figure 26) correspondant à l'exécution de T_2 puis de T_1 en transformant l'exécution concurrente E_1 : par exemple, les opérations `read A de T2` et `write A de T1` sont conflictuelles. Cela montre bien que l'exécution concurrente E_1 et l'exécution en série E_3 ne sont pas équivalentes (et, donc que l'exécution en série E_2 n'est pas équivalente non plus à l'exécution en série E_3).

En disant autrement un point énoncé plus haut, pour montrer qu'une exécution concurrente est sérialisable, il suffit de montrer au moins une des exécutions en série équivalentes que l'on peut obtenir par une suite de permutations d'opérations non-conflictuelles. Pour montrer qu'une exécution concurrente n'est PAS sérialisable, il faut montrer que l'on ne peut obtenir AUCUNE des exécutions en série que l'on aurait pu lui associer par une suite de permutations d'opérations non-conflictuelles.



Remarque

D'autres notations sont parfois utilisées :

- $r_i(D)$: la transaction T_i réalise une opération de lecture op_{Lect} de la donnée D ,
- $w_i(D)$: la transaction T_i réalise une opération d'écriture op_{Ecrit} de la donnée D ,
- On ne tient pas compte du tout des opérations op_{NoBD} sans accès à la BD.

L'exécution E_1 (cf. Figure 24) peut être notée :

$r_1(A) \quad w_1(A) \quad r_2(A) \quad r_1(B) \quad w_2(A) \quad w_1(B) \quad r_2(B) \quad w_2(B)$

Certaines personnes préfèrent travailler depuis une telle notation plutôt que depuis le « tableau » présentant l'exécution concurrente. Faites bien comme vous voulez : les 2 sont équivalents pour analyser la sérialisabilité d'une exécution concurrente.

Ainsi, dans le cadre du contrôle de concurrence, le gestionnaire transactionnel a la charge de surveiller l'exécution concurrente de transactions afin de voir si elle est sérialisable ou non. Charge à lui, si jamais il détecte un souci de sérialisabilité (donc de conflictualité donc de perte de la propriété d'isolation), de mettre en œuvre « ce qu'il faut » pour que l'exécution concurrente reste sérialisable... Il doit donc :

- Tester la sérialisabilité de l'exécution concurrente,
- Remédier à la non-sérialisabilité si besoin (*i.e.* si elle est détectée).

3.4.1.4 Détection de la non-sérialisabilité : graphe de précédence

Pour tester si une exécution concurrente est sérialisable ou non, le gestionnaire transactionnel construit en mémoire un graphe appelé **graphe de précédence**.



Définition : « graphe de précédence »

Un **graphe de précédence** pour une exécution concurrente E est un graphe orienté dans lequel :

- Les nœuds représentent les transactions de l'exécution concurrente E ,
- Il y a un arc orienté d'un nœud T_i vers un nœud T_j s'il existe dans E une opération op_i de T_i conflictuelle avec une opération op_j de T_j ET que op_i apparaît dans l'exécution concurrente E avant op_j ,
- Cet arc est étiqueté avec la donnée D sur laquelle portent les opérations conflictuelles op_i et op_j .



Exemple

Soit l'exécution concurrente E suivante :

$E = r_2(A) \quad r_1(B) \quad w_2(A) \quad r_3(A) \quad w_1(B) \quad w_3(A) \quad w_2(B)$

Son graphe de précédence est le suivant :

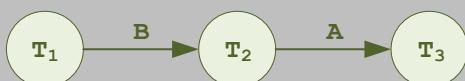


Figure 27. Exemple de graphe de précédence

L'intérêt principal de ces graphes de précédence est d'indiquer :

- Si l'exécution concurrente associée est sérialisable (le graphe de précédence ne comportant pas de cycle), à quelle(s) exécution(s) en série des mêmes transactions elle est équivalente,
- Si l'exécution concurrente associée n'est pas sérialisable (le graphe de précédence comportant au moins un cycle), quelles transactions « empêchent » sa sérialisabilité.

Tout vient du fait que la présence d'un arc a en réalité une vraie sémantique. Ainsi, dans le graphe de précédence d'une exécution concurrente E , la présence d'un arc depuis une transaction T_1 vers une transaction T_2 étiqueté par la donnée D signifie en réalité que s'il existe une (des) exécution(s) en série équivalente(s) à l'exécution concurrente E (et, donc, si elle est sérialisable) alors toutes ces exécutions en série équivalentes sont telles qu'elles exécutent toutes les opérations faisant partie de la transaction T_1 forcément avant²⁹ d'exécuter toutes les opérations faisant partie de la transaction T_2 . On appelle cela une **contrainte de précédence** (d'où le nom de « graphe de précédence »). Donc, un graphe de précédence définit ainsi un système de contraintes de précédences et :

- Si l'ensemble des contraintes de précédence exprimée par un graphe de précédence peuvent être respectée par une (des) exécution(s) en série des mêmes transactions, c'est que cette (ces) exécution(s) en série est (sont) équivalente(s) à l'exécution concurrente E et, puisqu'elle a au moins une exécution en série qui lui est équivalente, c'est donc qu'elle est sérialisable (et, donc, qu'elle respecte la propriété d'isolation).
- Si cet ensemble de contraintes de précédence ne peut pas être respecté, c'est qu'il n'existe aucune exécution en série des mêmes transactions qui puisse être équivalente à l'exécution concurrente E et, donc, qu'elle n'est pas sérialisable (et ne respecte donc pas la propriété d'isolation).

Exemple (début)

Reprendons l'exemple de l'exécution concurrente E suivante :

$E = r_2(A) \ r_1(B) \ w_2(A) \ r_3(A) \ w_1(B) \ w_3(A) \ w_2(B)$

On l'a vu, son graphe de précédence est le suivant :

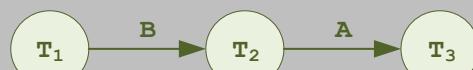


Figure 28. Rappel de l'exemple de graphe de précédence



Ce graphe de précédence contient 2 arcs :

1. De T_1 vers T_2 étiqueté par la donnée B ,
2. De T_2 vers T_3 étiqueté par la donnée A .

Ces 2 arcs ont la sémantique suivante : s'il existe une (des) exécution(s) en série équivalente(s) à l'exécution concurrente E_1 , alors, dans cette (ces) exécution(s) en série, on doit forcément respecter les contraintes suivantes...

1. Toutes les opérations de la transaction T_1 doivent précéder celles de T_2 ,
2. Toutes les opérations de la transaction T_2 doivent précéder celles de T_3 .

²⁹ Peu importe que ce soit « immédiatement avant » ou « bien avant ».

Exemple (fin)

L'exécution concurrente E_1 contient 3 transactions. En théorie, on peut donc « former » 6 (3!) exécutions en série de ces 3 transactions... Il faut donc chercher s'il en existe au moins une qui respecte les 2 contraintes de précédence énoncées ci-dessus :

- $E_{1S1} : T_1/T_2/T_3 \Rightarrow \text{OK}$, les 2 contraintes de précédence sont respectées,
- $E_{1S2} : T_1/T_3/T_2 \Rightarrow \text{NOK}$, T_2 n'est pas exécutée avant T_3 ,
- $E_{1S3} : T_2/T_1/T_3 \Rightarrow \text{NOK}$, T_1 n'est pas exécutée avant T_2 ,
- $E_{1S4} : T_2/T_3/T_1 \Rightarrow \text{NOK}$, T_1 n'est pas exécutée avant T_2 ,
- $E_{1S5} : T_3/T_1/T_2 \Rightarrow \text{NOK}$, T_2 n'est pas exécutée avant T_3 ,
- $E_{1S6} : T_3/T_2/T_1 \Rightarrow \text{NOK}$, les 2 contraintes de précédence sont violées.

Il existe donc bien une exécution en série équivalente à l'exécution concurrente E_1 (dans ce cas, une et une seule : E_{1S1}). Donc, cette exécution concurrente est sérialisable (et respecte donc la propriété d'isolation).

Soit une autre exécution concurrente E_2 :

$E_2 = r_2(A) \ r_1(B) \ w_2(A) \ r_2(B) \ r_3(A) \ w_1(B) \ w_3(A) \ w_2(B)$



Son graphe de précédence est le suivant :

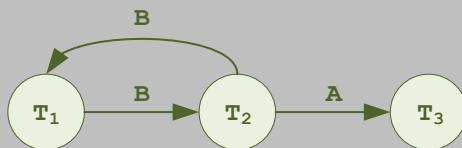


Figure 29. Autre exemple de graphe de précédence

Les contraintes de précédence sous-jacentes aux arcs de ce graphe de précédence indiquent que, dans toute exécution en série équivalente (s'il en existe), on doit avoir :

1. Toutes les opérations de la transaction T_1 doivent précéder celles de T_2 ,
2. Toutes les opérations de la transaction T_2 doivent précéder celles de T_1 ,
3. Toutes les opérations de la transaction T_2 doivent précéder celles de T_3 .

Il semble évident qu'un tel système de contraintes de précédence ne peut pas être respecté : il est impossible, dans une même exécution en série, de placer T_1 avant T_2 ET T_2 avant T_1 ! Donc, l'exécution concurrente E_2 n'est équivalente à aucune exécution en série des mêmes transactions : elle n'est donc pas sérialisable (et ne respecte pas la propriété d'isolation).

Les choses se passent en fait plus simplement que cela. On utilise toujours un graphe de précédence, mais en essayant de détecter directement sur le graphe si le système de contraintes de précédence qui « se cache » derrière est satisfaisable ou non. Il n'est donc pas question en réalité d'établir ce système de contraintes de précédence puis d'identifier chacune des exécutions en série que l'on pourrait former sur les mêmes transactions puis, pour chacune d'entre elles, de voir si elle respecte le système de contraintes de précédence !

Pour cela, on utilise l'observation suivante : finalement, pour qu'un système de contraintes de précédence soit « insoluble », il faut que 2 contraintes de précédence (ou plus) se « contredisent », directement ou indirectement (*i.e.* transitivement *via* d'autres contraintes de précédence du système). Au niveau du graphe de précédence, cela prend forcément la forme d'un cycle (direct ou indirect) dans les arcs de ce graphe. Ainsi, on peut affirmer que **si le graphe de précédence d'une exécution concurrente E ne contient PAS de cycle, alors cette exécution concurrente est sérialisable !** Forcément, *a contrario*, dès l'instant où un graphe de précédence comporte au moins un cycle, alors l'exécution concurrente qui lui est associée n'est PAS sérialisable.



Remarque

La manipulation de graphes (notamment la détection de cycles dans un graphe) est un problème complexe mais traité depuis très longtemps en informatique : on bénéficie donc de nos jours d'algorithme hyper performants pour cela. C'est la raison pour laquelle on utilise ici des graphes de précédence.



Attention

Pour la détection des cycles, n'oubliez pas que les graphes de précédence sont orientés.

De plus, toujours pour la détection des cycles, on ne tient PAS compte des données étiquetant les arcs : les contraintes de précédence ne portent pas sur les données des transactions, il n'y a pas de raison de les regarder quand on veut détecter un cycle dans le graphe de précédence !



Exemple

Le graphe de précédence de l'exécution concurrente E_1 (voir ci-dessous) ne contient aucun cycle, donc l'exécution concurrente E_1 est sérialisable. De plus, les arcs de ce graphe de précédence nous indiquent que, dans les exécutions en série équivalentes (s'il en existe), T_1 doit être avant T_2 ET T_2 avant T_3 . Il n'y a donc qu'une seule exécution en série des mêmes transactions qui soit équivalente à l'exécution concurrente E_1 : c'est $T_1/T_2/T_3$.

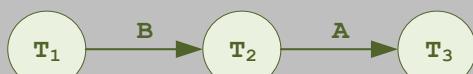


Figure 30. Graphe de précédence de E_1 ne contenant aucun cycle

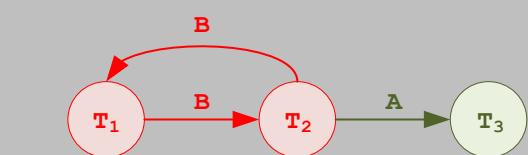


Figure 31. Graphe de précédence de E_2 contenant un cycle

Reprendons les exemples d'exécutions concurrentes donnés plus haut et donnant naissance aux 4 types de problèmes d'accès concurrentiels aux données et traçons les graphes de précédence associés pour constater l'application de ce que nous venons de voir...

- Pour la perte de mise à jour : le graphe de précédence associé contient un cycle, donc l'exécution concurrente n'est pas sérialisable (et ne respecte donc pas la propriété d'isolation).

E		Remarques
T ₁	T ₂	A = 10
1 start		1
2 read A in varA ₁		2 T ₁ lit A = 10
3	start	3
4	read A in varA ₂	4 T ₂ lit A = 10
5 varA ₁ := varA ₁ + 10		5
6 write A from varA ₁		6 T ₁ écrit A = 20
7	varA ₂ := varA ₂ + 50	7
8	write A from varA ₂	8 T ₂ écrit A = 60
9	9

Figure 32. Rappel de l'exemple de perte de mise à jour

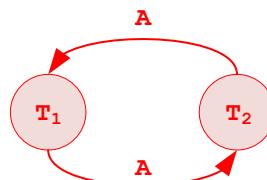


Figure 33. Graphe de précédence de l'exemple de perte de mise à jour

- Pour la lecture impropre de données incohérentes : le graphe de précédence associé contient, là encore, un cycle, donc l'exécution concurrente n'est pas sérialisable.

E		Remarques
T ₁	T ₂	A + B = 200 A = 120, B = 80
1 start		1
2 read A in varA ₁		2 T ₁ lit A = 120
3 varA ₁ := varA ₁ - 50		3
4 write A from varA ₁		4 T ₁ écrit A = 70
5	start	5
6	read A in varA ₂	6 T ₂ lit A = 70
7	read B in varB ₂	7 T ₂ lit B = 80
8	varC ₂ := varA ₂ + varB ₂	8
9	display varC ₂	9 Affiche 150 !
10	read B in varB ₁	10 T ₁ lit B = 80
11	varB ₁ := varB ₁ + 50	11
12	write B from varB ₁	12 T ₁ écrit B = 130
13	13

Figure 34. Rappel de l'exemple de lecture impropre (données incohérentes)

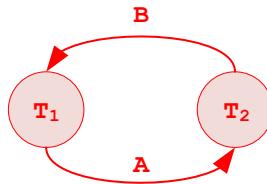


Figure 35. Graphe de précédence de l'exemple de données incohérentes

- Pour la lecture impropre de données non confirmées : le graphe de précédence associé contient, une fois de plus, un cycle, donc l'exécution concurrente n'est pas sérialisable.

E		Remarques
T ₁	T ₂	
1	start	1
2	varA ₂ := 70	2
3	write A from varA ₂	3 T ₂ écrit A = 70
4	start	4
5	read A in varA ₁	5 T ₁ lit A = 70
6	rollback	6 T ₂ écrit A = 50
7	...	7

Figure 36. Rappel de l'exemple de lecture impropre (données non confirmées)

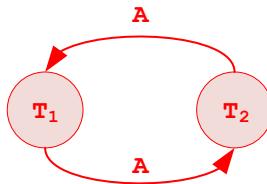


Figure 37. Graphe de précédence de l'exemple de données non confirmées

- Et enfin, pour la lecture non reproductible : une dernière fois, le graphe de précédence associé contient, un cycle, donc l'exécution concurrente n'est pas sérialisable.

E		Remarques
T ₁	T ₂	
1	start	1
2	read A in varA ₂	2 T ₂ lit A = 10
3	start	3
4	varA ₁ := 20	4
5	write A from varA ₁	5 T ₁ écrit A = 20
6	read A in varA ₂	6 T ₂ lit A = 20
7	...	7

Figure 38. Rappel de l'exemple de lecture non-reproductible

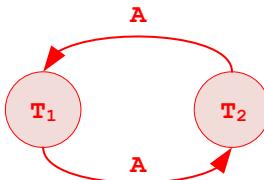


Figure 39. Graphe de précédence de l'exemple de lecture non reproductible

On sait donc détecter si une exécution concurrente présente des problèmes d'accès concurrentiels en raison du non-respect de la propriété d'isolation : il suffit de « tracer » son graphe de précédence et de voir s'il contient un cycle ou non...

3.4.1.5 Traitement en cas de non-sérialisabilité

Détecter, c'est une chose, mais « réparer », c'en est une autre !



Question

Comment « traiter » les problèmes d'accès concurrentiels une fois qu'ils ont été détectés ?

Réponse

Toujours grâce aux graphes de précédence.

Vous l'aurez compris, les problèmes d'accès concurrentiels sont causés par les transactions présentant de le ou les cycles (quand il y en a, bien sûr) du graphe de précédence (en fait, par des opérations conflictuelles au sein de ces transactions). Le principe de la résolution de ce problème est en fait simple³⁰ : l'idée est de « défaire » (annuler) une (ou plusieurs ou toutes) des transactions impliquées dans le cycle. Ainsi, plus de cycle, donc l'exécution concurrente redévient sérialisable et aucun problème d'accès concurrentiel aux données ne se pose plus puisque la propriété d'isolation est de nouveau respectée !

En pratique, quand il détecte un cycle dans le graphe de précédence, le gestionnaire transactionnel « insère » arbitrairement un rollback dans un (ou plusieurs ou toutes) des transactions impliquées dans le cycle. Ceci l'annule et « règle le problème »...



Question

Comment choisir quelle(s) transaction(s) du cycle annuler ?

Réponse

Toute l'éventuelle difficulté est là, cela dépend des capacités du gestionnaire transactionnel.

Un gestionnaire transactionnel « simple » va en annuler une au hasard (ou alors la première ou la dernière) alors qu'un gestionnaire transactionnel évolué va pouvoir aller jusqu'à analyser les traitements qu'elles ont déjà faits afin de procéder à l'annulation qui provoquera elle-même le moins d'annulations en cascade³¹.

³⁰ Contrairement à sa mise en œuvre parfois.

³¹ L'annulation d'une transaction T₁ peut nécessiter l'annulation d'une transaction T₂ (*etc.*), notamment si l'annulation de T₁ restaure des valeurs modifiées par T₁ et que T₂ a travaillé sur ces valeurs modifiées.

3.4.2 Verrouillage : mieux vaut prévenir que guérir

Une autre technique plus classique pour prévenir les problèmes dus à la concurrence est le **verrouillage** : chaque transaction verrouille les données qu'elle lit ou écrit pour interdire aux autres transactions d'y accéder³².

Vous allez le voir au travers des différents exemples, le verrouillage dégrade les performances d'un SGBD, notamment en imposant des temps d'attente. On peut limiter cette perte de performances en précisant la nature des opérations pour lesquelles le verrouillage est réalisé (par exemple lecture d'un côté et écriture d'un autre). Cela conduit à la définition de **modes de verrouillage** (cf. §3.4.2.1). On peut encore améliorer les performances en limitant la taille des données à verrouiller, d'où le concept de **granularité de verrouillage** (cf. §3.4.2.5).

Le principe général est le suivant :

1. Selon l'opération qu'elle souhaite réaliser sur une donnée D , une transaction demande à verrouiller cette donnée D (dans le mode le plus adapté à l'opération à réaliser et pour la granularité correspondant à la donnée à verrouiller),
2. Le gestionnaire transactionnel peut alors prendre 2 décisions, en tenant compte du mode du verrou demandé et des éventuels verrous déjà existants sur la donnée D ,
 - a. Il autorise la pose de verrou : la transaction se poursuit alors normalement,
 - b. Il ne peut autoriser la pose du verrou à ce moment-là : il met la transaction en attente jusqu'à ce que la pose du verrou puisse enfin être accordée.

3.4.2.1 Modes de verrouillage

Les modes de verrouillage varient d'un SGBD à l'autre. Deux modes de verrouillage sont cependant toujours définis :

- *Partagé (noté S)* : demandé afin de lire une donnée,
- *Exclusif (noté X)* : demandé afin de modifier une donnée.

Les règles qui régissent ces deux modes sont les suivantes :

- Un verrou partagé (S) ne peut être obtenu sur une donnée que si les verrous déjà placés sur cette donnée sont eux-mêmes partagés (S) ou si la donnée n'est pas verrouillée,
- Un verrou exclusif (X) ne peut être obtenu sur une donnée que si aucun verrou n'est déjà placé sur cette donnée.

Usuellement, ces règles, qui régissent donc les modes de verrouillage, sont exprimées au moyen d'une **matrice de compatibilité**. Pour les modes S et X , cette matrice de compatibilité est la suivante :

Matrice de compatibilité MC des verrous en mode S et X		Verrou(s) déjà posé(s) sur la donnée		
		Aucun	Tous en mode S	Au moins un en mode X
Verrou demandé sur la donnée...	En mode S	Oui	Oui	Attente
	En mode X	Oui	Attente	Attente

Tableau 14. Matrice de compatibilité des verrous en mode S et X

³² Cette technique de verrouillage existe dans bien d'autres contextes que les SGBD, par exemple en Java avec les méthodes `wait()`, `notify()` et `notifyAll()` de la classe `Object`, notamment utilisées en programmation parallèle afin d'éviter les problèmes d'interférences entre processus légers (*threads*) ! 😊

Dans cette matrice de compatibilité MC, MC (i, j) indique si un verrou en mode i peut être obtenu sur une donnée quand un verrou en mode j y est déjà placé. Si c'est le cas, la transaction ayant demandé à poser ce verrou se poursuit, sinon elle est mise en attente.



Remarque

En regardant bien cette matrice, on retrouve les règles de définition de la conflictualité entre opérations transactionnelles (cf. §3.4.1.3) : 2 opérations sont conflictuelles si elles opèrent sur la même donnée ET si au moins l'une d'entre elles est une opération d'écriture. C'est exactement ce qu'on retrouve ici en considérant les opérations de lecture ou d'écriture qui vont suivre les demandes de verrouillage (en fonction du mode demandé pour le verrou).

Les deux événements transactionnels suivants gèrent le verrouillage et s'ajoutent aux 5 événements transactionnels d'une transaction vus précédemment³³ :

Événement	Signification
lock m D	Demande d'un verrou en mode m sur la donnée D
unlock D	Déverrouillage d'une donnée D

Tableau 15. Événements transactionnels de verrouillage/déverrouillage

Par suite d'un événement transactionnel lock m D , il va donc être possible de voir que la transaction ayant fait cette demande de verrouillage a été placée en attente au sein de l'exécution concurrente par le gestionnaire transactionnel (parce que cette demande ne pouvait pas être satisfaite au moment-même de sa demande, en raison de la matrice de compatibilité).



Exemple

L'exécution concurrente suivante fait bien apparaître les demandes de verrouillage, les opérations de déverrouillage et les éventuelles mises en attente...

E		
	T ₁	T ₂
1	start	
2	lock S A	
3	read A in varA ₁	
4	varB ₁ := varA ₁ * 2	
5		start
6		lock S B
7		read B in varB ₂
8	lock X B	
9	Attente...	varC ₂ := varB ₂ - 1
10	Attente...	lock X C
11	Attente...	write C from varC ₂
12	Attente...	unlock B
13	write B from varB ₁	
14

Figure 40. Exemple d'utilisation des événements de verrouillage/déverrouillage

³³ Pour mémoire, les 5 événements transactionnels déjà vus sont start, read A in varA, write A from varA, commit et rollback.

En rebouclant sur ce qu'on a déjà dit sur le verrouillage, lorsqu'une transaction T obtient un verrou en mode m sur une donnée D :

- Si c'est un verrou en mode partagé (S), la transaction T a des droits en lecture uniquement sur la donnée D jusqu'à ce qu'elle déverrouille cette donnée,
- Si c'est un verrou en mode exclusif (X), la transaction T a des droits en lecture ET en écriture sur la donnée D jusqu'à ce qu'elle déverrouille cette donnée.

Remarque

On pourrait alors se dire qu'il suffit de demander à verrouiller toujours en mode exclusif X et le problème est réglé : on peut en effet aussi bien lire qu'écrire la donnée verrouillée !



Mais non : il ne faut pas oublier que le verrouillage d'une donnée fait perdre de la performance (ce sont les mises en attente). Il est donc inutile de verrouiller en mode exclusif X une donnée à un moment où on a juste besoin de la lire, surtout que cela interdit d'autres transactions de faire n'importe quel accès à la donnée ! Si, au moment de cette lecture on la verrouille uniquement en mode partagé S , on autorise d'autres transactions à venir lire cette même donnée, ce qui favorise l'accès concurrentiel et, donc, amoindrit la perte de performances due au verrouillage.

Pour favoriser la concurrence durant une exécution concurrente de transactions qui utilisent les opérations de verrouillage et de déverrouillage, il faut *a priori* :

- Demander les verrous le plus tard possible, *i.e.* juste avant d'avoir besoin de travailler dessus,
- Demander les verrous les plus « fins » possible : en mode partagé S si on a juste besoin de lire la donnée, en mode exclusif X si on a besoin de l'écrire.
- Déverrouiller les données le plus tôt possible, *i.e.* juste après avoir fini de travailler dessus.

Exemple

L'exécution concurrente suivante ne respecte PAS les principes énoncés ci-dessus favorisant l'accès concurrentiel : ici, on demande dès le début des verrous en mode exclusif X sur toutes les données manipulées et on ne déverrouille qu'à la fin... **À NE PAS FAIRE, DONC !**



E		
	T_1	T_2
1	start	
2	lock X A	
3	lock X B	
4	read A in varA ₁	
5	varB ₁ := varA ₁ * 2	
6		start
7		lock X B
8	write B from varB ₁	Attente...
...	...	Attente...
...	unlock A	Attente...
...	unlock B	Attente...
...		lock X C
...	commit	...

Figure 41. Exemple de verrouillage dégradant les performances

Le bon usage (si possible le plus performant possible) de cette technique du verrouillage permet donc de « protéger » ponctuellement (*i.e.* localement au moment où elle est verrouillée) une donnée D d'accès faits par d'autres transactions et pouvant interférer.



Question

Donc, le bon usage de la technique du verrouillage assure le respect de la propriété d'isolation et de la propriété de cohérence ?

Réponse

Pas tout à fait : cela ne va pas suffire tout seul 😞 L'usage du verrouillage est nécessaire mais pas suffisant...



Exemple

L'exécution concurrente ci-dessous utilise bien la technique du verrouillage :

- Les données sont verrouillées le plus tard et le plus finement possible,
- Elles sont déverrouillées le plus tôt possible.

Néanmoins, un problème d'accès concurrentiel se pose toujours (donc la propriété d'isolation n'est pas respectée) : on a encore un problème de lecture non-reproductible !!!

E		Remarques
T ₁	T ₂	
1	start	1
2	lock S A	2
3	read A in varA ₂	3 T ₂ lit A = 10
4	start	4
5	varA ₁ := 20	5
6	lock X A	6
7	Attente...	7
8	unlock A	8
9	write A from varA ₁	9 T ₁ écrit A = 20
10	lock S A	10
11	unlock A	11 T ₂ lit A = 20
12	...	12

Figure 42. Exemple de lecture non-reproductible malgré les verrouillages

Afin d'éviter les problèmes d'accès concurrentiels (et, donc, retrouver le respect de la propriété d'isolation), il va falloir compléter l'usage de la technique du verrouillage par d'autres mécanismes... Le plus connu et probablement le plus courant est celui du « verrouillage à deux phases »...

3.4.2.2 Verrouillage à deux phases

Le mécanisme du verrouillage à deux phases vient compléter l'usage des modes de verrouillage vu ci-dessus. Ce mécanisme repose en fait sur un compromis³⁴ : on est prêt à perdre en peu en performances (un peu plus que dans l'exemple ci-dessus) mais, en contrepartie, on peut garantir le respect de la propriété d'isolation !

³⁴ Une fois de plus !!! 😞

3.4.2.2.1 Technique du verrouillage à deux phases

La technique du verrouillage à deux phases repose sur la définition de « catégories » de transactions (la 2^{nde} définition reposant en fait sur la 1^{ère}) :



Définition : « transaction bien formée »

Une transaction est **bien formée** si elle respecte les 2 propriétés suivantes :

- Elle obtient un verrou sur une donnée avant de lire ou de modifier cette donnée,
- Elle libère tous ses verrous avant de se terminer.



Définition : « transaction à deux phases »

Une transaction est **à deux phases** si elle respecte les 2 propriétés suivantes :

- Elle est bien formée,
- Après sa première opération de libération de verrou (*i.e.* de déverrouillage), de n'importe quelle donnée, elle n'acquiert plus AUCUN verrou, sur n'importe quelle donnée.



En pratique

Autrement dit, une transaction est à deux phases si elle est bien formée, donc, et si sa première opération de déverrouillage `unlock` n'est suivie d'AUCUNE opération de demande de verrouillage `lock` ! De plus, il ne faut pas oublier que les transactions sont écrites indépendamment les unes des autres, donc sans savoir comment leurs opérations « s'inséreront » parmi les opérations d'autres transactions dans une même exécution concurrente...

C'est à cause de cette dernière propriété à respecter que l'on parle de transactions à deux phases. On distingue en effet :

1. *Une phase d'acquisition des verrous* : elle s'étend du démarrage `start` de la transaction (inclus) jusqu'à l'opération (inclus) qui précède son premier déverrouillage `unlock`,
2. *Une phase de libération des verrous* : elle s'étend de la première opération de déverrouillage `unlock` (inclus) jusqu'à sa terminaison (`commit` ou `rollback`) inclus.



Exemple

La transaction ci-dessous est bien une transaction à deux phases :

T	
1	<code>start</code>
2	<code>lock X A</code>
3	<code>read A in varA</code>
4	<code>lock S B</code>
5	<code>read B in varB</code>
6	<code>unlock B</code>
7	<code>varA := varA + varB</code>
8	<code>write A from varA</code>
9	<code>unlock A</code>
10	<code>commit</code>

Figure 43. Exemple de transaction à 2 phases

Afin de favoriser l'accès concurrentiel aux données, le protocole habituel consiste à lever les verrous lors de la fin de la transaction (`commit` ou `rollback`). Les opérations transactionnelles `commit` et `rollback` sont donc « modifiées » de façon à lever automatiquement tous les verrous de la transaction terminées qui n'auraient pas préalablement été explicitement levés au moyen de l'opération `unlock`³⁵. Ainsi, en cas « d'oubli » de déverrouillage, tous les verrous restants demandés par une transaction sont levés à la fin de cette transaction (confirmation ou annulation).

**Question**

Pourquoi faire des opérations de déverrouillage `unlock` alors ?

Réponse

Toujours dans le même objectif : pour favoriser l'accès concurrentiel aux données !

Il convient de lever explicitement un verrou (`unlock`) lorsque l'on sait qu'il sera inutile pour la poursuite de la transaction (de façon à favoriser la concurrence).

**Question**

OK, je sais écrire des transactions à deux phases, et alors ?

Réponse

Alors c'est bon : vous avez tout gagné ! 😊🍾

3.4.2.2.2 Verrouillage à deux phases et sérialisabilité

Il est démontré qu'une exécution concurrente d'un ensemble de transactions qui sont TOUTES à deux phases est forcément sérialisable. En conséquence, il ne peut y avoir ni pertes de mise à jour, ni lectures impropres, ni lectures non reproductibles lorsque l'on considère une exécution ne contenant que des transactions à deux phases. En bref, la propriété d'isolation est forcément respectée dans une exécution concurrente de transactions qui sont toutes à 2 phases.

**Remarque**

On ne fera pas la démonstration ici, mais elle est liée au fait que la matrice de compatibilité rappelle les règles de conflictualité entre opérations transactionnelles et au fait qu'on ne demande pas de nouveau verrou après avoir commencé à en libérer.

Reprendons encore une fois les exemples d'exécutions concurrentes donnés plus haut et donnant naissance aux 4 types de problèmes d'accès concurrentiels aux données. Réécrivons ces exemples sous la forme d'exécutions concurrentes de transaction à 2 phases (T2P) et traçons aussi les graphes de précédence associés pour constater l'application de ce que nous venons de voir...

- Pour la perte de mise à jour : le verrouillage de A par la transaction T_1 oblige la transaction T_2 à attendre que la transaction T_1 ait terminé sa mise à jour de A avant d'entamer la sienne. Il n'y a donc pas (plus) de perte de mise à jour.

³⁵ De plus, pour faire les éventuelles écritures de restauration de valeurs qu'elle a à faire, l'opération d'annulation `rollback` fait toute seule et correctement le demandes de déverrouillage nécessaires si les verrous utiles ne sont plus « présents » et, bien sûr, les déverrouillages associés.

E		Remarques	
	T ₁	T ₂	A = 10
1	start		1
2	lock X A		2
3	read A in varA ₁		3 T ₁ lit A = 10
4		start	4
5		lock X A	5
6	varA ₁ := varA ₁ + 10	Attente...	6
7	write A from varA ₁	Attente...	7 T ₁ écrit A = 20
8	unlock A	Attente...	8
9		read A in varA ₂	9 T ₂ lit A = 20
10		varA ₂ := varA ₂ + 50	10
11		write A from varA ₂	11 T ₂ écrit A = 70
12		unlock A	12
13	13

Figure 44. Exemple de perte de mise à jour réécrit avec des T2P



Figure 45. Graphe de précédence de l'exemple de perte de mise à jour (T2P)

- Pour la lecture impropre de données incohérentes : le verrouillage de A par la transaction T₁ empêche la transaction T₂ de lire A avant que la transaction T₁ n'ait terminé sa mise à jour de A et de B. La transaction T₂ affiche donc une valeur cohérente de A + B soit 200.

E		Remarques	
	T ₁	T ₂	A + B = 200 A = 120, B = 80
1	start		1
2	lock X A		2
3	read A in varA ₁		3 T ₁ lit A = 120
4	varA ₁ := varA ₁ - 50		4
5	write A from varA ₁		5 T ₁ écrit A = 70
6		start	6
7		lock S A	7
8	lock X B	Attente...	8
9	read B in varB ₁	Attente...	9 T ₁ lit B = 80
10	varB ₁ := varB ₁ + 50	Attente...	10
11	write B from varB ₁	Attente...	11 T ₁ écrit B = 130
12	unlock A	Attente...	12
13		read A in varA ₂	13 T ₂ lit A = 70
14	unlock B		14
15		lock S B	15
16		read B in varB ₂	16 T ₂ lit B = 130
17		varC ₂ := varA ₂ + varB ₂	17
18		display varC ₂	18 Affiche 200
19	19

Figure 46. Exemple de lecture impropre (données incohérentes) réécrit avec des T2P



Figure 47. Graphe de précédence de l'exemple de lecture impropre (données incohérentes) (T2P)

- Pour la lecture impropre de données non confirmées : le verrouillage de A par la transaction T_2 empêche la transaction T_1 de lire A avant que la transaction T_2 n'ait été annulée. La transaction T_1 lit donc bien la valeur qu'avait A avant que la transaction T_2 ne commence.

E		Remarques
	T ₁	T ₂
1		start 1
2		lock X A 2
3		varA ₂ := 70 3
4		write A from varA ₂ 4 T_2 écrit A = 70
5	start	
6	lock S A	
7	Attente...	rollback 7 T_2 écrit A = 50 A déverrouillé
8	read A in varA ₁	
9	...	

Figure 48. Exemple de lecture impropre (données non confirmées) réécrit avec des T2P

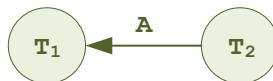


Figure 49. Graphe de précédence de l'exemple lecture impropre (données non confirmées) (T2P)

- Et enfin, pour la lecture non reproductible : le verrouillage de A par la transaction T_2 durant le temps de ses deux lectures empêche la transaction T_1 de modifier A entre ces deux lectures. La transaction T_2 lit donc bien deux fois la même valeur de A soit 10.

E		Remarques
	T ₁	T ₂
1		start 1
2		lock S A 2
3		read A in varA ₂ 3 T_2 lit A = 10
4	start	
5	lock X A	
6	Attente...	read A in varA ₂ 6 T_2 lit A = 10
7	Attente...	unlock A 7
8	varA ₁ := 20	
9	write A from varA ₁	
10	...	

Figure 50. Exemple de lecture non reproductible réécrit avec des T2P

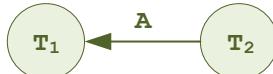


Figure 51. Graphe de précédence de l'exemple lecture non reproductible (T2P)

Puisqu'une exécution concurrente de transactions qui sont toutes à 2 phases est forcément sérialisable, elle est donc équivalente à au moins une exécution en série des mêmes transactions. On sait que l'ordre de sérialisation (*i.e.* l'ordre des transactions dans au moins une des exécutions en série équivalentes) est celui des instants d'acquisition du dernier verrou (*i.e.* du dernier `lock` de chaque transaction à 2 phases de l'exécution concurrente).



Exemple

Ainsi, dans les 4 cas précédents, on sait qu'on a équivalence entre l'exécution concurrentes de transactions à 2 phases et au moins l'exécution en série suivante :

- Pour la perte de mise à jour : exécution en série T₁ puis T₂,
- Pour la lecture impropre (données incohérentes) : T₁ puis T₂ également,
- Pour la lecture impropre (données non confirmées) : T₂ puis T₁,
- Pour la lecture non reproductible : T₂ puis T₁ également.

3.4.2.3 Interblocage

L'inconvénient de la technique du verrouillage est le risque d'**interblocage** (ou **verrou mortel** ou **deadlock** en anglais) entre plusieurs transactions.



Définition : « interblocage (d'un ensemble de transactions) »

L'interblocage d'un ensemble de transactions se produit quand toutes ces transactions s'attendent les unes les autres pour pouvoir se poursuivre. Chaque transaction concernée par l'interblocage est donc dans une position d'attente d'un déverrouillage par une autre transaction de l'interblocage, elle-même en attente.



Exemple

L'exécution concurrente ci-dessous illustre la situation d'interblocage entre 2 transactions :

E		
	T ₁	T ₂
1	start	
2	lock X A	
3	varA ₁ := 10	
4		start
5		varB ₁ := 50
6		lock X B
7	lock S B	
8	Attente...	lock S A
9	Attente...	Attente...
10	Attente...	Attente...
11	Attente...	Attente...
12

Figure 52. Exemple d'interblocage

On sent intuitivement bien le souci que peut poser l'arrivée d'un interblocage ! Heureusement, le SGBD est capable de détecter et de traiter cette situation...

3.4.2.3.1 Détection d'un interblocage : graphe d'attente

Pour détecter un interblocage, le SGBD élabore en mémoire un **graphe d'attente**.



Définition : « graphe d'attente »

Un **graphe d'attente** pour une exécution concurrente E est un graphe orienté dans lequel :

- Les nœuds représentent les transactions de l'exécution concurrente E ,
- Il y a un orienté arc d'un nœud T_i vers un nœud T_j si, dans l'exécution concurrente E , la transaction T_i attend que la transaction T_j libère son verrou sur la donnée D pour elle-même y placer un verrou en mode m ,
- Cet arc est étiqueté par $D(m)$ où D est la donnée « cause » de l'attente et m est le mode de verrouillage que la transaction T_i à l'origine de l'arc orienté demande pour verrouiller la donnée D .



Attention

Ne confondez PAS graphes de précédence et graphes d'attente !!! Un graphe de précédence indique si une exécution de transactions concurrentes est sérialisable ou non alors qu'un graphe d'attente indique si des transactions sont en attente de verrous.

Une situation d'interblocage se traduit par l'apparition d'un cycle dans le **graphe d'attente** de l'exécution concurrente.



Exemple

Le graphe d'attente de l'exécution concurrente précédente (cf. Figure 52) est le suivant :

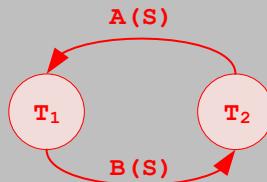


Figure 53. Exemple de graphe d'attente montrant un interblocage

Il contient un cycle qui confirme l'interblocage.



Attention

Comme pour les graphes de précédence, pour la détection des cycles, n'oubliez pas que les graphes d'attente sont orientés. De plus, toujours pour la détection des cycles, on ne tient PAS compte des étiquettes sur les arcs.

Exemples

En reprenant les 4 exemples d'exécutions concurrentes de transactions à 2 phases, les graphes d'attente associés sont les suivants :

- Pour la perte de mise à jour (cf. Figure 44) :



Figure 54. Graphe d'attente de la perte de mise à jour réécrite en T2P

- Pour la lecture impropre de données incohérentes (cf. Figure 46) :



Figure 55. Graphe d'attente de la lecture impropre de données incohérentes réécrite en T2P

- Pour la lecture impropre de données non confirmées (cf. Figure 48) :



Figure 56. Graphe d'attente de la lecture impropre de données non confirmées réécrite en T2P

- Pour la lecture non reproductible (cf. Figure 50) :



Figure 57. Graphe d'attente de la lecture non reproductible réécrite en T2P

3.4.2.3.2 Traitement de l'interblocage

Deux techniques peuvent être utilisées pour résoudre l'interblocage :

- La détection : on laisse les situations d'interblocage se produire et on inspecte à intervalles réguliers le graphe d'attente pour détecter si un interblocage s'est produit. Dans ce cas, on défait arbitrairement (rollback) l'une des transactions bloquées (*i.e.* présente dans le cycle au sein du graphe d'attente) et on la relance un peu plus tard.

Question

Comment choisir la transaction à annuler ?

Réponse

Un gestionnaire transactionnel « simple » va en annuler une au hasard (ou alors la première ou la dernière) alors qu'un gestionnaire transactionnel évolué va pouvoir aller jusqu'à analyser les traitements qu'elles ont déjà faits afin de procéder à l'annulation qui provoquera elle-même le moins d'annulations en cascade³⁶.

³⁶ L'annulation d'une transaction T₁ peut nécessiter l'annulation d'une transaction T₂ (*etc.*), notamment si l'annulation de T₁ restaure des valeurs modifiées par T₁ et que T₂ a travaillé sur ces valeurs modifiées.

- *La prévention* : on adopte en fait un protocole de travail qui évite de façon certaine les verrous mortels !
 - Par exemple en accordant à une transaction tous les verrous dont elle a besoin dès son démarrage (juste après start en fait),
 - Ou bien en fixant un ordre arbitraire sur les données de la BD et en imposant aux transactions de respecter cet ordre dans leurs demandes de verrous.

Stratégie de traitement de l'interblocage	Avantages	Inconvénients
...par détection	Simplicité de traitement lorsque l'interblocage survient	Nécessité de surveiller si un interblocage d'attente
...par prévention	L'interblocage n'arrive jamais	Perte de performances (par limitation des possibilités d'accès concurrentiels)

Tableau 16. Avantages et inconvénients des stratégies de traitement de l'interblocage

Les interblocages étant relativement rares, la technique de détection est souvent plus économique.

3.4.2.4 Reclassement d'un verrou

Nous venons de le voir, l'écriture de transactions à 2 phases impose une contrainte sur l'écriture des transactions, aucune demande de verrouillage ne pouvant être faite après la première opération de déverrouillage. Cette contrainte, bien que simple à respecter, amoindrit les performances (puisque elle réduit les possibilités d'accès concurrentiels).



Exemple

Soit une transaction qui lit puis modifie une donnée D. Si elle fonctionne en verrouillage à deux phases, elle devra demander un verrou en mode exclusif X dès la lecture de D. Il n'y aura donc pas de concurrence possible entre la lecture de D et son déverrouillage après sa mise à jour.

T	Remarques
1 start	
2 lock X A	
3 read A in varA	Aucun accès concurrentiel à la donnée A n'est possible sur toute cette « plage » d'opérations
4 ...	
5 write A from varA	
6 unlock A	
7 ...	

Figure 58. Exemple de transaction à deux phases amoindrisant les accès concurrentiels

Afin d'augmenter le degré de concurrence, on introduit la possibilité de **reclasser** un verrou...



Définition : « reclassement de verrou »

Le **reclassement de verrou** permet à une transaction possédant déjà un verrou sur une donnée D de modifier le mode m de verrouillage associé (on peut donc modifier ce mode de S en X ou de X en S).

La technique du reclassement est mise en œuvre via deux nouvelles opérations transactionnelles :

- *Le surclassement* : il permet à une transaction possédant déjà un verrou en mode partagé S sur une donnée D de demander à passer ce verrou en mode exclusif X,
- *Le déclassement* : il permet à une transaction possédant déjà un verrou en mode exclusif X sur une donnée D de faire passer ce verrou en mode partagé S.



Remarque

Il est important de noter que :

- Le surclassement réduisant les possibilités d'accès concurrentiels à la donnée D, il doit être demandé par la transaction : c'est le gestionnaire transactionnel qui décide d'accorder le surclassement (auquel cas la transaction se poursuit) ou de mettre cette demande en attente (jusqu'à ce qu'elle puisse être satisfaite),
- Le déclassement augmentant les possibilités d'accès concurrentiels à la donnée D, il n'est pas besoin de le demander : il est réalisé immédiatement !

Les deux nouveaux événements transactionnels correspondants sont les suivants :

Événement	Signification
upgrade D	Demande de surclassement (en mode exclusif X) du verrou en mode partagé S déjà possédé par la transaction sur la donnée D
downgrade D	Déclassement (en mode partagé S) du verrou en mode exclusif X déjà possédé par la transaction sur la donnée D

Tableau 17. Événements transactionnels de reclassement

Les règles régissant l'obtention d'un surclassement sont les mêmes que les règles d'obtention d'un verrou en mode X (cf. Tableau 14).

La définition de ces opérations de surclassement/déclassement modifie les propriétés des modes de verrouillage dans les SGBD qui les prennent en charge :

- *Possibilités d'accès (inchangées) en mode partagé S* : une transaction ayant posé un verrou sur une donnée en mode partagé S a des droits en lecture uniquement sur cette donnée,
- *Possibilités d'accès (modifiées) en mode exclusif X* : une transaction ayant posé un verrou sur une donnée en mode exclusif X a des droits en écriture uniquement (**mais plus en lecture**) sur cette donnée.



Question

Pourquoi une telle restriction en mode exclusif X ?

Réponse

Afin de favoriser l'accès concurrentiel aux données, ici en « forçant » à ne pas être en mode exclusif (dans lequel les accès concurrentiels sont restreints) si aucune écriture n'est réalisée.



Exemple

La transaction ci-dessous utilise les opérations de reclassement de verrous :

	T		Remarques
1	start	1	
2	lock S A	2	Accès concurrentiels à la donnée A possibles
3	read A in varA	3	en lecture
4	varA := varA + 10	4	Accès concurrentiels à la donnée A
5	upgrade A	5	impossibles
6	write A from varA	6	
7	downgrade A	7	
8	...	8	

Figure 59. Exemple d'utilisation des opérations de reclassement

En plus de la restriction des possibilités de traitement sur une donnée quand elle est verrouillée en mode exclusif X, l'introduction des opérations de reclassement a d'autres impacts, dont les plus importants sont les suivants :

- Une transaction est à deux phases si elle est bien formée et si aucune opération de verrouillage (lock) **ni de surclassement (upgrade)** ne suit sa première opération de déverrouillage unlock **ou de déclassement (downgrade)** :
 - *Phase d'acquisition* : lock et upgrade,
 - *Phase de libération* : unlock et downgrade.
- De façon générale, pour augmenter le degré de concurrence d'une exécution concurrente de transactions qui utilisent les opérations de verrouillage, de déverrouillage et de reclassement, il faut *a priori* :
 - Demander le plus tard possible les verrous (inchangé),
 - Surclasser le plus tard possible les verrous,
 - Déclasser le plus tôt possible les verrous,
 - Déverrouiller le plus tôt possible les données (inchangé).

Les opérations de fin de transaction commit et rollback déverrouillent toujours automatiquement tous les verrous qui n'auraient pas été explicitement levés par la transaction terminée.



En pratique

Rappelons que les transactions sont écrites indépendamment les unes des autres, donc sans savoir comment leurs opérations « s'inséreront » parmi les opérations d'autres transactions dans une même exécution concurrente...

3.4.2.5 Verrouillage à granularité multiple

Jusqu'à maintenant, quand nous avons abordé les opérations liées à la consultation (read), à la modification (write et, selon le cas, commit ou rollback) et à la gestion de verrous (lock, unlock, upgrade, downgrade, commit et rollback), nous avons parlé de « donnée » sans préciser la nature de cette donnée. En fait, ce terme peut, dans l'absolu dans une BD, désigner aussi bien une valeur d'un attribut d'un n-uplet donné, un n-uplet entier, mais aussi une table voire toute une BD ! On parle alors de **niveau de granularité** (ou, plus simplement, de **granularité**) de la donnée manipulée. Cette notion prend tout son sens dès qu'on réfléchit aux manipulations (y compris usuelles) que l'on est amenés à faire sur une BD...



Exemple

Par exemple, une requête qui calcule la moyenne des âges d'un ensemble de personnes doit pouvoir verrouiller cet ensemble en lecture durant le temps de calcul en interdisant toute modification d'un sous-ensemble de cet ensemble.

Toujours en vue d'optimiser les possibilités d'accès concurrentiels aux données (donc les performances), un SGBD peut offrir un mécanisme de verrouillage permettant de tenir compte de ces différents niveaux de granularité : c'est le **protocole de verrouillage à granularité multiple**. Ce protocole permet de placer des verrous sur des objets de granularité variable imbriqués hiérarchiquement.



Question

Pourquoi « imbriqués hiérarchiquement » ?

Réponse

Il se trouve que, dans une BD relationnelle, les niveaux de granularité identifiés théoriquement sont liés par une relation d'imbrication :

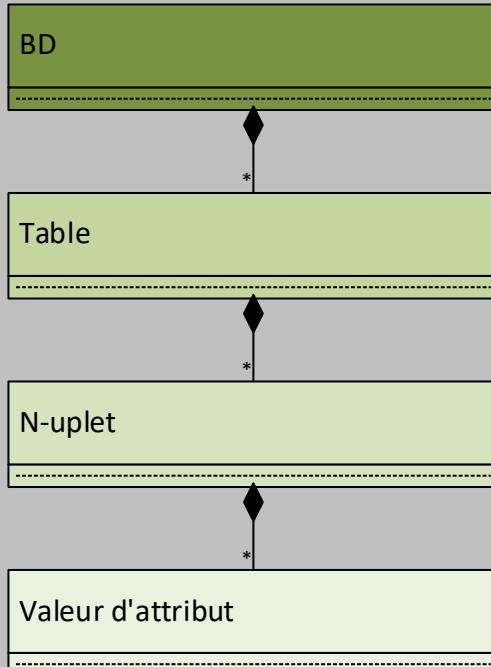


Figure 60. Niveaux de granularité « théoriques » dans une BD



Attention

Dans la réalité, certains SGBD ne gèrent qu'une partie de ces niveaux de verrouillage (*i.e.* parfois seulement 2 ou 3 d'entre eux³⁷)... Mais cela ne change rien à ce qui va être dit ci-après : le protocole de verrouillage à granularité multiple s'applique de la même façon !

³⁷ Et seule la documentation du SGBD pourra alors vous indiquer si c'est le cas et, si oui, quels sont les niveaux de granularité effectivement pris en charge.

Le protocole de verrouillage à granularité multiple rajoute trois modes de verrouillage dits « intentionnels » qui lui sont propres (en plus des modes partagé S et exclusif X) :

- **Le mode d'intention partagée (IS)** : une transaction demande à verrouiller un objet O en mode d'intention partagée IS lorsqu'elle souhaite lire (et uniquement lire) des sous-objets de O ,
- **Le mode d'intention exclusive (IX)** : une transaction demande à verrouiller un objet O en mode d'intention exclusive IX lorsqu'elle souhaite modifier des sous-objets de O ,
- **Le mode d'intention partagée exclusive (SIX)** : une transaction demande à verrouiller un objet O en mode d'intention partagée exclusive SIX lorsqu'elle souhaite modifier des sous-objets de O , mais que cette modification nécessite une lecture préalable de sous-objets de O (pas forcément les mêmes, voire de tout l'objet O).

En pratique

L'exemple le plus simple de besoin de verrouillage en mode d'intention partagée exclusive SIX est l'usage de l'opérateur UPDATE de SQL :

```
UPDATE Livre
SET Prix := 1.25 * Prix
WHERE Catégorie = 'Roman';
```



Le traitement de l'instruction de mise à jour ci-dessus va effectivement modifier des sous-objets de la table Livre (en l'occurrence la valeur de l'attribut Prix de certains n-uplets de cette table) mais, pour ce faire, a besoin de lire des sous-objets de la même table (en l'occurrence la valeur de l'attribut Catégorie de tous ses n-uplets).

La mise en œuvre du protocole de verrouillage intentionnel est régie de la façon suivante...

En pratique

Le pseudo-algorithme de verrouillage intentionnel est le suivant :



```
Pseudo-algorithme VerrouillageIntentionnel
Données d'entrée
 $O_{final}$  : un objet à lire ou écrire
Données de sortie
Aucune (mais les verrous adéquats sont posés)
Données intermédiaires
 $O_{courant}$  : l'objet sur lequel on est positionné
Début
 $O_{courant} \leftarrow$  objet racine de la hiérarchie à
Laquelle appartient  $O_{final}$ 
Tant que ( $O_{courant} \neq O_{final}$ ) faire
    Verrouiller  $O_{courant}$  en mode IS si  $O_{final}$  est à
    verrouiller en mode S ou verrouiller  $O_{courant}$ 
    en mode IX ou SIX si  $O_{final}$  est à verrouiller
    en mode X
     $O_{courant} \leftarrow$  descente vers  $O_{final}$  d'un niveau
    dans la hiérarchie contenant  $O_{final}$ 
FinTQ
    Verrouiller  $O_{courant}$  en mode S pour le lire
    ou en mode X pour l'écrire
Fin
```

Comme pour les modes de verrouillage « classiques » (*i.e.* partagé S et exclusif X), la pose de verrous dans l'un de ces modes est demandée par une transaction au gestionnaire transactionnel (toujours *via* l'opération lock) et le déverrouillage est acté dès qu'il apparaît (toujours *via* l'opération unlock). Les règles d'accord des demandes de verrouillage sont exprimées là encore au moyen d'une matrice de compatibilité :

Matrice de compatibilité MC des verrous « classiques » et des verrous intentionnels	Verrou(s) déjà posé(s) sur la donnée						
	Aucun	En mode IS	En mode IX	En mode SIX	En mode S	En mode X	
Verrou demandé sur la donnée...	En mode IS	Oui	Oui	Oui	Oui	Oui	Attente
	En mode IX	Oui	Oui	Oui	Attente	Attente	Attente
	En mode SIX	Oui	Oui	Attente	Attente	Attente	Attente
	En mode S	Oui	Oui	Attente	Attente	Oui	Attente
	En mode X	Oui	Attente	Attente	Attente	Attente	Attente

Tableau 18. Matrice de compatibilité des verrous « classiques » ET intentionnels



Remarques

Soit un objet t contenu dans un objet o :

- Si on veut lire t alors il faut empêcher toute modification de o dans sa totalité : pour lire t, il faut le verrouiller en mode S, ce qui implique de verrouiller o en mode IS, ce qui interdit de verrouiller o en mode X et donc de modifier o.
- Si on veut modifier t alors il faut empêcher toute lecture ou modification de o dans sa totalité : pour modifier t, il faut le verrouiller en mode X, ce qui implique de verrouiller o en mode IX ou SIX, ce qui interdit de verrouiller o en mode S ou X et donc de lire ou modifier o dans sa totalité.
- Le verrouillage en mode IX (resp. IS) de o par une transaction T₂ est accepté même s'il a déjà été verrouillé en mode IS (resp. IX) par une transaction T₁ car il se peut que le sous-ensemble des n-uplets lus (resp. modifiés) par T₁ soit disjoint des n-uplets modifiés (resp. lus) par T₂. Dans le cas contraire, le verrouillage des n-uplets communs en mode S ou X empêcherait le conflit.



Exemple de requête nécessitant un verrouillage en mode SIX

Reprendons la requête de mise à jour R appartenant à la transaction T :

```
UPDATE Livre
SET Prix := 1.25 * Prix
WHERE Catégorie = 'Roman';
```

Pour réaliser cette mise à jour, la transaction T dans laquelle figure la requête R peut vouloir un accès exclusif à la table Livre pour empêcher toute modification de cette table durant l'opération de mise à jour. Si T verrouille la table Livre en mode X, elle ne pourra pas lire les n-uplets de Livre pour sélectionner ceux qui doivent être mis à jour. Cependant, si T verrouille la table Livre en mode SIX (accès partagé exclusif) alors elle et elle seule pourra verrouiller les n-uplets de cette table en mode S pour les lire afin de sélectionner ceux qui sont des romans qui, eux, seront ensuite verrouillés en mode X afin d'être modifiés.

Un intérêt secondaire du protocole de verrouillage à granularité multiple, c'est qu'il permet de résoudre le problème des **objets fantômes**.



En pratique

Ce type de problème est assimilable à une lecture non reproductible : deux transactions (ou plus) s'exécutent en parallèle et l'une « voit apparaître » un objet (par exemple un n-uplet) alors que ça n'est pas elle qui l'a inséré. La propriété d'isolation n'est alors pas respectée !



Exemple (début)

Soit la relation Livre (Titre, Année). Considérons l'exécution concurrente ci-dessous :

	E		Remarques
	T ₁	T ₂	
1	start		1
2	SELECT count(*) FROM Livres l WHERE l.Année = 2003;		2 T ₁ lit n
3		start	3
4		INSERT INTO Livres VALUES ("Les BD", 2003);	4
5	SELECT count(*) FROM Livres l WHERE l.Année = 2003;		5 T ₁ lit n+1
6	6

Figure 61. Exemple de problème d'objet fantôme

Le problème est le suivant : sans verrouillage, on a une lecture non reproductible. En effet, la transaction T₁ n'a pas vu l'ajout du n-uplet ("Les BD", 2003) ajouté par la transaction T₂. Pour elle, ce n-uplet est donc un n-uplet fantôme.

La solution est d'utiliser le protocole de verrouillage intentionnel. Mais si on considère uniquement les niveaux « n-uplet » et « valeur d'attribut », on ne résout pas le problème car il est impossible de verrouiller un objet qui n'existe pas (le n-uplet inséré par T₂ ne peut pas être verrouillé avant son insertion !). En revanche, si on considère en plus au moins le niveau « table », le verrouillage à granularité multiple résout le problème par un verrouillage en mode IS de la table Livre (et le déverrouillage associé, bien sûr)...

Exemple (fin)

Supposons donc que l'on connaît les niveaux de granularité « n-uplet » et « table » (puisque nous faut ici au moins le second). L'exécution concurrente ci-dessous reprend la précédente (cf. Figure 61) en rajoutant l'usage du protocole de verrouillage intentionnel :

	E		Remarques
	T ₁	T ₂	
1	start		1
2	lock IS Livres;		2
3	SELECT count(*) FROM Livres l WHERE l.Année = 2003;		3 T ₁ lit n
4		start	4
5		lock X Livres;	5
6	SELECT count(*) FROM Livres l WHERE l.Année = 2003;	Attente...	6 T ₁ lit n
7	unlock Livres;	Attente...	7
8		INSERT INTO Livres VALUES ("Les BD", 2003);	8
9	9

Figure 62. Exemple de problème d'objet fantôme résolu grâce au verrouillage intentionnel

Question

L'exemple ci-dessus montre bien l'usage de verrous intentionnels. Mais où sont passées les demandes de verrouillage en mode partagé S et/ou exclusif X ?

Réponse

Elles n'ont pas disparu : elles sont en fait la plupart du temps demandées implicitement par le SGBD (mais pas forcément de façon optimale) !

Ainsi, dans le dernier exemple (cf. Figure 62), considérons la « requête » suivante :

```
SELECT count(*)
FROM Livre l
WHERE l.Année = 2003;
```

Rappel

Rappelons que, dans ce cadre, l désigne tour à tour chaque n-uplet de la relation Livres (c'est un itérateur sur les n-uplets de la table).

Toujours en supposant que l'on connaît les niveaux de granularité « n-uplet » et « table », ce n-uplet l est automatiquement verrouillé en mode partagé S pour le lire (en fait pour lire la valeur de son attribut Année).

Toujours dans le même exemple (cf. Figure 62), considérons cette autre « requête » :

```
INSERT INTO Livres
VALUES ("Les BD", 2003);
```

En fait, le verrouillage en mode exclusif X de la table `Livres` est lui aussi demandé automatiquement si cela n'a pas été explicitement fait (on aurait donc pu ne pas le demander explicitement ci-dessus).



En pratique

Les verrous en mode « classique » (*i.e.* partagé S ou exclusif X) peuvent parfois être demandés implicitement par le SGBD³⁸ s'ils n'ont pas été explicitement demandés. Cela dit, ces demandes « automatiques » ne sont pas forcément faites de façon optimale : mieux vaut donc, quand c'est possible (*i.e.* pas au sein d'opérations complexes), les faire explicitement « au meilleur endroit » !

En revanche, les verrous en mode intentionnel ne sont JAMAIS demandés implicitement : ils doivent toujours être demandés explicitement (d'où leur nom).

3.5 Paramètres du gestionnaire transactionnel

Les paramètres caractérisant un gestionnaire transactionnel sont les suivants :

Notation	Unité(s) usuelle(s)	Paramètre
-	-	Mode de mise à jour de la base de données : immédiate ou différée .
-	-	Type de transactions : non-contraintes , à 2 phases ou « spéciales ».
-	-	Opérations de reclassement de verrous supportées ? Non , Partiellement (voir les « limitations imposées » ci-dessous), Totalement .
-	-	Stratégie de résolution des problèmes d'accès concurrentiels : simple ou évoluée .
-	-	Stratégie de résolution des problèmes d'interblocage : détection ou prévention .
-	-	Limitations supplémentaires imposées : des limitations particulières peuvent être imposées et sont alors décrites ici.

Tableau 19. Paramètres du gestionnaire transactionnel

³⁸ Bien sûr, cela dépend de votre SGBD : seule sa documentation vous dira s'il prend cela en charge ou non. C'est cependant forcément réalisé pendant la réalisation des opérations complexes (SFW, INSERT, UPDATE, ALTER, ...).

4 Gestionnaire de reprise après panne

SOMMAIRE DÉTAILLÉ DU CHAPITRE 4

4.1	Types de pannes	76
4.2	Reprise « intuitive »	77
4.3	Annulation de transactions en cascade	80
4.4	Annulations en cascade : traitement « intuitif » et « évitement »	81
4.5	Surveillance des transactions : journalisation	82
4.5.1	Tenue du journal de reprise après panne	82
4.5.2	Point de reprise	84
4.6	Reprise après panne : utilisation du journal de reprise	85
4.6.1	Classification des transactions après une panne	85
4.6.2	Journalisation et reprise en mode de mise à jour immédiate	86
4.6.2.1	Déroulement d'une transaction et journalisation	86
4.6.2.2	Reprise après panne	86
4.6.3	Journalisation et reprise en mode de mise à jour différée	88
4.6.3.1	Déroulement d'une transaction et journalisation	88
4.6.3.2	Reprise après panne	88
4.7	Paramètres de la journalisation	88

FIGURES DU CHAPITRE 4

Figure 66.	Exemple de panne pendant une exécution concurrente	78
Figure 67.	Exemple de panne pendant une exécution avec accès concurrentiels	80
Figure 68.	Exemple de tenue de journal de reprise après panne	83
Figure 69.	Exemple d'exécution concurrente à journaliser	87
Figure 70.	Exemple de journal de reprise (en mode de mise à jour immédiate)	87
Figure 71.	Exemple de journal de reprise (en mode de mise à jour différée)	88

TABLEAUX DU CHAPITRE 4

Tableau 20.	Événements consignés dans le journal de reprise	83
-------------	---	----

ÉQUATIONS DU CHAPITRE 4

Aucune entrée de table d'illustration n'a été trouvée.

Encore plus que pour d'autres logiciels, la panne est l'ennemi du SGBD : le temps de redémarrage (qui peut être long pour de très grosses BD) est forcément source de pertes (à plusieurs niveaux) mais, et c'est le point le plus prégnant, il y a de forts risques qu'elle entraîne une perte de données !

4.1 Types de pannes

Les pannes d'un SGBD peuvent être classées en trois catégories :

- **Abandon de transaction (rollback)** : il peut survenir à la suite d'une erreur de programme, d'un abandon explicite de l'utilisateur ou être provoqué par le gestionnaire de transactions,



En pratique

Bien que l'abandon de transaction soit théoriquement considéré comme une « panne », ça n'en est pas une au sens classique du terme : le gestionnaire transactionnel gère ces abandons très bien puisque l'opération rollback qui en est la source fait partie des opérations basiques qu'il doit savoir gérer (surtout si les accès concurrentiels sont contrôlés par des dispositifs de verrouillage) ! On ne traitera donc pas dans ce chapitre ce type de « panne ».

- **Panne logicielle ou matérielle (hors mémoire de stockage)** : les modifications enregistrées en mémoire de travail et non propagées en mémoire de stockage sont perdues ; en revanche, les données déjà enregistrées en mémoire de stockage sont conservées,



En pratique

Puisque les données enregistrées en mémoire de stockage sont conservées, on pourrait penser que « tout va bien » et qu'il n'y a qu'à redémarrer le SGBD et recharger ces données... Mais non !!! En effet, les modifications faites par des transactions sont propagées dans la BD³⁹ via le dispositif de mémoire cache. Nous étudierons la mémoire cache plus loin (cf. §7) mais sachez d'ores et déjà que cela peut être assimilé à un sous-ensemble « virtuel » de la BD. Il est stocké en mémoire vive et est « virtuel » en ce sens qu'il contient bien les informations de la BD dans leur version la plus récente MAIS qu'il n'est pas synchronisé en temps réel avec la BD telle qu'elle est stockée en mémoire stable. Rien n'assure donc que, au moment de la panne, les modifications ont été transférées de la mémoire cache vers la mémoire de stockage. **Il est donc fort probable que les données enregistrées en mémoire de stockage au moment de la panne soient celles de la BD alors qu'elle est dans un état incohérent !** Impossible donc de juste recharger ces données pour tout relancer... Nous allons donc étudier ici ce type de panne.

- **Panne de la mémoire de stockage** : tout ou partie des données enregistrées sur mémoire de stockage sont perdues.



En pratique

Seuls des mécanismes de duplication (sauvegarde, redondance, ...) peuvent permettre de retrouver les données perdues. Ces techniques de duplication usuelles ne sont pas propres aux SGBD⁴⁰, nous ne les traiterons donc pas ici.

³⁹ Immédiatement si on est en mode de mise à jour immédiate ou à la confirmation (commit) si on est en mode de mise à jour différée.

⁴⁰ Même si certains les mettent fortement en avant, parfois en les adaptant.

4.2 Reprise « intuitive »

Comme annoncé ci-dessus, nous ne traiterons donc « que » des pannes logicielles ou matérielles (hors mémoire de stockage). Face à une telle panne, il est facilement possible d'avoir une « intuition » de ce qu'il faut faire pour remettre la BD dans un état cohérent :

1. Relancer le SGBD (éventuellement après avoir remplacé le matériel défectueux et, donc, relancé l'ordinateur),
2. Remettre la BD (telle qu'elle est enregistrée en mémoire de stockage) dans un état cohérent,
3. Permettre la mise en œuvre de nouveaux traitements sur la BD.



En pratique

On parle de « reprise à chaud ».

Le point essentiel dans ces étapes est, bien sûr, la remise de la BD dans un état cohérent. Bien sûr, cette « remise en état cohérent » doit respecter les propriétés transactionnelles ACID.



Question

Pourquoi la BD serait-elle dans un état incohérent au moment de la panne ?

Réponse

On l'a déjà dit, tout vient du fait que les modifications sont propagées *via* la mémoire cache.

Ainsi, en considérant les 2 modes de propagation des modifications transactionnelles dans la BD (modes de mise à jour) :

- *Si on est en mode de mise à jour immédiate* : toute modification est immédiatement propagée dans la BD, mais *via* la mémoire cache. Rien ne permet donc d'assurer que les propagations réalisées par une transaction confirmée (*commit*) n'ont été propagées depuis la mémoire cache vers la mémoire de stockage au moment de la panne. Rien ne permet non plus d'assurer que les modifications réalisées par une transaction annulée (*rollback*) avant son annulation ne sont pas encore présentes sur la mémoire de stockage (les réécritures aux valeurs initiales se faisant elles aussi *via* la mémoire cache) !
- *Si on est en mode de mise à jour différée* : les modifications sont propagées dans la BD uniquement au moment de la confirmation (*commit*) de la transaction les ayant réalisées. Mais, là encore, ces propagations sont faites *via* la mémoire cache : on n'est donc jamais certain que les modifications demandées par une transaction confirmée aient été propagées en mémoire de stockage. En revanche, dans ce mode de mise à jour, on est certain qu'une transaction annulée ou pas encore confirmée n'a pu propager aucune modification en mémoire de stockage !



En pratique

Comme cela est indiqué plus haut, nous étudierons la mémoire cache plus loin (cf. §7) mais sachez d'ores et déjà que cela peut être assimilé à un sous-ensemble « virtuel » de la BD. Il est stocké en mémoire vive et est « virtuel » en ce sens qu'il contient bien les informations de la BD dans leur version la plus récente MAIS qu'il n'est pas synchronisé en temps réel avec la BD telle qu'elle est stockée en mémoire stable.

En pratique

Précisément, les sources possibles de problèmes sont liées à la propagation des modifications en mémoire de stockage *MS* et sont les suivantes :

- *En mode de mise à jour immédiate :*
 - Une transaction en cours a pu propager des modifications en *MS*,
 - Une transaction confirmée a pu propager une partie seulement de ses modifications en *MS*,
 - Une transaction annulée a pu propager une partie seulement des annulations de modifications en *MS*.
- *En mode de mise à jour différée :*
 - Une transaction en cours n'a propagé aucune modification en *MS*,
 - Une transaction confirmée a pu propager une partie seulement de ses modifications en *MS*,
 - Une transaction annulée n'a propagé aucune modification en mémoire de stockage, elle n'a donc aucune annulation de modification à réaliser.



On sent que le traitement de la reprise sera un peu plus simple si le SGBD est en mode de mise à jour différée. Mais, le travail sera, dans le principe, le même.

C'est donc bien le dispositif de mémoire cache qui pose ici un problème. Cela dit, étant donnés les gains de performances qu'il permet, il est hors de question de s'en passer. Il faut donc trouver un moyen d'annuler ces « effets de bord » indésirables dus à son usage... Étudions pour cela un exemple d'exécution concurrente au cours de laquelle une panne survient.

Exemple

Soit l'exécution concurrente E durant laquelle une panne survient :

E			Remarques
	T ₁	T ₂	
1	start		1
2	lock S A		2
3	read A in varA ₁		3 T ₁ lit A = 15
4	varA ₁ := varA ₁ - 1		4
5	upgrade A		5
6	write A from varA ₁		6 T ₁ écrit A = 14
7	lock S B		7
8	unlock A		8
9		start	9
10		lock S C	10
11		read C in varC ₂	11 T ₂ lit C = 50
12		varC ₂ := varC ₂ * 2	12
13		upgrade C	13
14		write C from varC ₂	14 T ₂ écrit C = 100
15		commit	15
16	read B in varB ₁		16 T ₁ lit B = 2
PANNE			

Figure 63. Exemple de panne pendant une exécution concurrente



Intuitivement, on sent bien que :

- *Si on est en mode de mise à jour immédiate* : il faut annuler les modifications faites par les transactions non confirmées au moment de la panne (étant donné qu'elles ont peut-être déjà été propagées en mémoire de stockage) ET il faut réécrire les modifications faites par les transactions déjà confirmées au moment de la panne (étant donné qu'elles n'ont pas forcément été propagées en mémoire de stockage),

Question

Annuler des modifications !? Mais on va perdre des traitements effectués !!!



Réponse

Oui, c'est vrai ! Mais ces traitements font partie d'un tout : la transaction. Et le principe d'atomicité indique bien que tous les traitements d'une même transaction doivent être effectués ou aucun. Il est donc logique que si certains des traitements d'une transaction ont été effectués avant la panne, mais pas tous, alors il faut annuler ceux qui ont été faits. Et il ne faut pas oublier que seule une opération de confirmation *commit* indique la bonne fin d'une transaction, donc l'accomplissement de tous les traitements qu'elle contient !

De plus, il est nettement préférable de remettre la BD dans un état cohérent quitte à « perdre » des traitements plutôt que l'inverse ! Les traitements « perdus » pourront toujours être relancés...

- *Si on est en mode de mise à jour différée* : il faut juste réécrire les modifications faites par les transactions déjà confirmées au moment de la panne (étant donné qu'elles n'ont pas forcément été propagées en mémoire de stockage) MAIS on n'a pas besoin d'annuler des modifications faites par des transactions non confirmées au moment de la panne (étant donné que ces modifications n'ont jamais été propagées du tout).



Exemple

Ainsi, en reprenant l'exemple ci-dessus (cf. Figure 63) :

- *Si on est en mode de mise à jour immédiate* :
 - La transaction T_1 n'ayant pas été confirmée au moment de la panne, il faut annuler la modification qu'elle a faite sur la donnée A (celle-ci ayant pu être propagée en mémoire de stockage) et en restaurer la valeur 15,
 - La transaction T_2 ayant été confirmée au moment de la panne, il faut refaire la modification qu'elle a faite sur la donnée C (celle-ci ayant pu ne PAS être propagée en mémoire de stockage) et en réécrire la valeur 100.
- *Si on est en mode de mise à jour différée* : seule la réécriture de la valeur 100 pour la donnée C est à réaliser.

Question

Et si on réécrit une valeur déjà propagée en mémoire de stockage ?



Réponse

Aucune importance ! En restant sur le même exemple, si la nouvelle valeur de la donnée C n'a pas été propagée en mémoire de stockage, on écrase la « mauvaise » valeur (50) par la « bonne » (100). En revanche, si la nouvelle valeur de la donnée C a été propagée en mémoire de stockage, on écrase la « bonne » valeur (100) par la « bonne » (100). Cela ne perturbe donc rien ! On fait peut-être une réécriture pour rien mais, étant donné qu'on est dans l'incertitude, mieux vaut faire cette réécriture parfois pour rien plutôt que de risquer de relancer la BD dans un état incohérent !

À ce point, on a une bonne première idée de ce qu'il faut faire : refaire toutes les modifications censées être pérennes (puisque'on doute de leur propagation en mémoire de stockage au moment de la panne) ET annuler toutes les modifications censées ne pas avoir été faites (puisque'on doute là encore de leur propagation en mémoire de stockage au moment de la panne).

4.3 Annulation de transactions en cascade

Cela dit, le problème n'est pas aussi simple : l'exemple précédent (cf. Figure 63) montre une exécution concurrente de 2 transactions, certes, mais ces transactions n'opèrent pas sur les mêmes données. Il n'y a donc aucun réel accès concurrentiel aux données dans ce cas. Mais, bien sûr, il faut se pencher sur le cas de réelles exécutions de transactions avec accès concurrentiels aux données (ce qui, de plus, est le cas le plus fréquent, et de loin !). Reprenons un dérivé de l'exemple précédent...

Exemple (début)

Soit l'exécution concurrente E' durant laquelle une panne survient :

E'			Remarques
	T ₁	T ₂	
1	start		1
2	lock S A		2
3	read A in varA ₁		3 T ₁ lit A = 15
4	varA ₁ := varA ₁ - 1		4
5	upgrade A		5
6	write A from varA ₁		6 T ₁ écrit A = 14
7	lock S B		7
8	unlock A		8
9		start	9
10		lock S A	10
11		read A in varA ₂	11 T ₂ lit A = 14
12		varA ₂ := varA ₂ * 2	12
13		upgrade A	13
14		write A from varA ₂	14 T ₂ écrit A = 28
15		commit	15
16	read B in varB ₁		16 T ₁ lit B = 2
PANNE			

Figure 64. Exemple de panne pendant une exécution avec accès concurrentiels



Exemple (fin)

Si on suit ce que l'on a énoncé ci-dessus, il faudrait :

- Annuler la modification faite par la transaction T_1 sur la donnée A (puisque la transaction T_1 était non confirmée au moment de la panne),
- Refaire la modification faite par la transaction T_2 sur la donnée A (puisque la transaction T_2 était, elle, confirmée au moment de la panne).

Mais, la modification faite par la transaction T_2 dépendant d'une valeur de la donnée A lue APRÈS la modification faite par la transaction T_1 , l'annulation de la modification faite par la transaction T_1 DOIT logiquement annuler aussi le travail de T_2 sur cette donnée A donc toute la transaction T_2 ⁴¹. Si d'autres transactions concurrentes avaient travaillé sur une valeur de la donnée A récupérée entre le moment de sa modification par la transaction T_1 et la panne, il aurait fallu également les annuler en cascade... tout en n'oubliant pas que leur propre annulation, aussi bien que celle de la transaction T_2 , en cascade de celle de la transaction T_1 peut à leur tour entraîner d'autres annulations en cascade !

4.4 Annulations en cascade : traitement « intuitif » et « évitement »

Toujours « intuitivement », on sent bien que, pour bien faire une reprise après panne en remettant la BD dans un état cohérent, il faut surveiller (AVANT la panne bien sûr, donc pendant l'exécution des transactions) :

- *Le début des transactions (start)* : il est nécessaire de savoir quelle transaction a été exécutée avant la panne (qu'elle soit confirmée ou non à ce moment-là),
- *La confirmation des transactions (commit)* : afin de repérer celles pour lesquelles il faudra faire des annulations (en mode de mise à jour immédiate uniquement) et celles pour lesquelles il faudra refaire les modifications (quel que soit le mode de mise à jour),
- *Les opérations de lecture (read)* : afin de repérer quelles données ont été lues par quelles transactions afin de pouvoir faire des annulations en cascade si nécessaire,
- *Les opérations d'écriture (write ou commit ou rollback selon le mode de mise à jour)* : afin de repérer quelles modifications ont été faites pour pouvoir les annuler ou les refaire si besoin.

La surveillance de ces opérations devrait être sauvegardée de façon pérenne afin d'être utilisée pour remettre la BD dans un état cohérent après la panne...

Il est cependant possible d'adopter une hypothèse simplificatrice afin d'éviter d'avoir à défaire les transactions en cascade : les verrous posés par une transaction ne sont libérés QUE lors de sa confirmation (*commit*). Avec un tel « protocole » d'écriture des transactions (sans opération *unlock*, donc), on n'a pas besoin de garder une trace des opérations de lecture des données (ce qui aurait été nécessaire pour repérer les éventuelles transactions à défaire en cascade), donc de surveiller ces opérations de lecture.

⁴¹ Par respect du principe d'atomicité, on ne peut pas défaire une partie d'une transaction : toute annulation « partielle » est en fait réalisée par une annulation complète !

**Remarque**

On perd alors bien sûr en possibilités d'accès concurrentiels, donc en performances, mais on est non seulement certain de n'avoir que des transactions à deux phases (ce qui a l'avantage de régler les problèmes d'accès concurrentiels, cf. §3.4.2.2.2) mais aussi de ne jamais avoir à défaire des transactions en cascade lors de la reprise.

Dans le cadre de la reprise après panne⁴², nous allons adopter cette hypothèse simplificatrice. On n'a donc *a priori* besoin de « surveiller » uniquement les opérations de démarrage, de confirmation et d'écriture réalisées par les transactions ! 😊

4.5 Surveillance des transactions : journalisation

La bonne nouvelle est que notre intuition était la bonne ! Le SGBD se charge, *via* un service particulier appelé **gestionnaire de reprise** (après panne) de réaliser cette surveillance et, en cas de redémarrage après une panne, d'utiliser cela pour remettre la BD dans un état cohérent. Le gestionnaire de reprise utilise une technique simplissime⁴³ : celle de la **journalisation**.

**Définition**

La journalisation consiste à tracer, dans un fichier journal enregistré sur une mémoire de stockage pérenne, un ensemble d'événements afin, en cas de besoin, de retrouver la trace de ces événements.

Ainsi, dans notre contexte, le **journal de reprise** (après panne) est un fichier séquentiel enregistré durablement sur une mémoire de stockage pérenne et qui contient une suite d'événements concernant la vie des transactions, dont notamment les modifications faites depuis un instant donné.

4.5.1 Tenue du journal de reprise après panne

En tenant compte de ce que nous avons dit plus haut⁴⁴, on va donc garder la trace dans un journal de reprise des opérations de démarrage, de confirmation et d'écriture faites par les transactions. Afin de simplifier l'utilisation du journal de reprise pour remettre la BD dans un état cohérent après une panne, nous introduisons également ce que nous appelons la pose de **points de reprise** (mais nous allons y revenir, cf. §4.5.2). Les traces qui seront conservées dans un journal de reprise sont donc les suivantes :

**En pratique**

Le journal de reprise est enregistré sur une **mémoire stable**, c'est-à-dire une mémoire qui, théoriquement, ne peut pas être détruite : par exemple sur des disques RAID (*Redundant Array of Inexpensive Disks*) en associant à chaque disque logique deux disques physiques dont l'un est le miroir de l'autre (ou toute autre technique de *RAID* plus évoluée).

⁴² Et UNIQUEMENT pour ce chapitre !

⁴³ Et maintenant fort répandue, notamment *via* la tenue de « fichiers logs » par pas mal de logiciels.

⁴⁴ Et de l'hypothèse simplificatrice adoptée pour ne pas avoir à défaire des transactions en cascade.

Événement	Marque posée dans le journal	Signification
Démarrage	(T "start")	La transaction T débute...
Écriture	En mode de MAJ immédiate : (T D a n)	La transaction T a modifié la valeur a de la donnée D vers la valeur n...
	En mode de MAJ différée : (T D n)	La transaction T a modifié la valeur de la donnée D et celle-ci vaut maintenant n...
Confirmation	(T "commit")	La transaction T est confirmée...
Point de reprise	("checkpoint")	Pose d'un point de reprise...

Tableau 20. Événements consignés dans le journal de reprise

On peut noter que l'écriture est consignée différemment selon le mode de mise à jour, c'est normal :

- *En mode de mise à jour immédiate* : on a vu qu'on peut être amené, lors de la reprise, à annuler des modifications faites par des transactions non confirmées (on a donc besoin de l'ancienne valeur des données modifiées) ainsi qu'à refaire des modifications faites par des transactions confirmées (on a donc besoin de la nouvelle valeur des données modifiées),
- *En mode de mise à jour différée* : on a vu qu'on n'a pas à annuler de transactions non confirmées mais juste à refaire des modifications faites par des transactions confirmées (on a donc besoin de la nouvelle valeur des données modifiées). Étant donné qu'on n'a PAS non plus à défaire de transactions en cascade (quel que soit le mode de mise à jour), on n'a pas besoin de garder trace des valeurs initiales des données modifiées dans ce mode de mise à jour.

Exemple

Soit l'exécution concurrente E. En mode de mise à jour immédiate, la portion du journal de reprise correspondant est le suivant (initialement A = 15, B = 2 et C = 20) :

E'		Journal de reprise
	T ₁	T ₂
1	start	1 (T1 "start")
2	lock S A	2
3	read A in varA ₁	3
4	varA ₁ := varA ₁ - 1	4
5	upgrade A	5
6	write A from varA ₁	6 (T1 A 15 14)
7	lock S B	7
9		9 (T2 "start")
10		10
11	lock S C	11
12	read C in varC ₂	12
13	varC ₂ := varC ₂ * 2	13
14	upgrade C	14 (T2 C 20 40)
15	write C from varC ₂	15 (T2 "commit")
16	commit	16
PANNE		

Figure 65. Exemple de tenue de journal de reprise après panne

4.5.2 Point de reprise

On le sent déjà venir : en cas de panne, le journal de reprise va être utilisé pour remettre la BD dans un état cohérent. Mais, la liste des traces transactionnelles qui y sont consignées peut vite devenir énorme⁴⁵. L'usage du journal pour la reprise après panne risque donc de vite être extrêmement coûteux.

On définit pour cela la notion de **point de reprise** : notamment afin de minimiser ce qu'on aura à utiliser dans le journal de reprise après une panne, donc d'optimiser le coût de cette reprise...



Définition : « point de reprise »

Un **point de reprise** est une « marque » écrite dans le journal de reprise qui indique qu'au moment où elle a été faite :

- Aucune transaction n'était en cours,
- Toutes les données écrites par des transactions antérieures au point de reprise avaient été transférées sur disque (mémoire stable).

Afin de respecter ces 2 conditions, pour obtenir un point de reprise, il faut :

- Afin de n'avoir aucune transaction en cours :
 - Interdire le début de nouvelles transactions,
 - Laisser se terminer les transactions en cours.
- Afin que toutes les écritures antérieures soient transférées sur mémoire de stockage : forcer la propagation des modifications présentes en mémoire cache vers la mémoire de stockage.



En pratique

Pour ce faire, le gestionnaire de reprise travaille « main dans la main » avec le gestionnaire transactionnel et avec le gestionnaire de mémoire cache. Tout se déroule comme suit...

De temps à autre (à intervalles plus ou moins réguliers et plus ou moins longs selon les SGBD), le gestionnaire de reprise après panne souhaite poser un point de reprise dans le journal. Pour ce faire :

1. Il en informe le gestionnaire transactionnel. Celui-ci laisse se finir les transactions en cours mais n'en démarre aucune nouvelle (leur démarrage est « mis en attente »). Forcément, à un moment, aucune transaction n'est plus en cours. Le gestionnaire transactionnel donne alors « son feu vert » au gestionnaire de reprise après panne.
2. Il informe alors le gestionnaire de mémoire cache. Celui-ci transfère sur mémoire de stockage toutes les pages contenues dans la mémoire cache et ayant été modifiées depuis qu'elles s'y trouvent. Ceci fait, toutes les écritures antérieures ont été propagées en mémoire de stockage : le gestionnaire de mémoire cache donne à son tour « son feu vert » au gestionnaire de reprise.
3. Il écrit la trace ("checkpoint") dans le journal de reprise.
4. Il force l'écriture de la dernière page du journal en mémoire de stockage.

⁴⁵ N'oubliez pas qu'on y conserve le démarrage et la confirmation de toutes les transactions, sans parler de toutes leurs opérations d'écriture (faites aussi bien par des write, commit ou rollback selon le mode de mise à jour) avec, pour chacune, au moins la nouvelle valeur, si ce n'est aussi la valeur initiale (en mode de mise à jour immédiate) !

4.6 Reprise après panne : utilisation du journal de reprise

Le journal de reprise après panne va pouvoir être utilisé pour remettre la BD dans un état cohérent au redémarrage suivant une panne.



Rappel

Rappelons qu'une écriture dans la BD se fait au travers de la mémoire cache. Une valeur écrite pourra donc résider dans la mémoire cache et n'être propagée en mémoire de stockage que bien plus tard. On n'est donc pas certain, lors de la reprise après panne, que les modifications effectuées avant la panne ont été transférées sur mémoire stable. Dans le doute, il faut donc :

- Refaire les modifications demandées par les transactions confirmées, quel que soit le mode de mise à jour,
- Annuler les modifications demandées par les transactions non confirmées, en mode de mise à jour immédiate uniquement.

4.6.1 Classification des transactions après une panne

Ainsi, lors de la reprise après la panne, on est amené à « classer » les transactions qui ont été « tracées » dans le journal de reprise. Ceci peut être fait en catégories :

- *Les transactions ayant été terminées avant le dernier point de reprise ("checkpoint")* : par définition-même de la pose d'un point de reprise, leurs modifications ont été enregistrées en mémoire de stockage. **Il n'y a donc rien à faire par rapport à ces transactions-là pendant la reprise après panne ! 😊**
- *Les transactions commencées après le dernier point de reprise ("checkpoint")* : **ce sont elles qu'il va donc falloir prendre en compte pendant la reprise après panne !** Ces transactions peuvent encore être classées en 2 sous-catégories...
 - *Les transactions commencées après le dernier point de reprise ("checkpoint") mais confirmées ("commit") avant la panne* : les nouvelles valeurs des données peuvent résider dans le tampon sans avoir été écrites sur mémoire stable. Il faut donc refaire dans la BD les modifications de ces transactions en réécrivant les nouvelles valeurs des données qui ont été mémorisées dans le journal.
 - *Les transactions commencées après le dernier point de reprise ("checkpoint") mais encore en cours au moment de la panne* : la propriété d'atomicité implique que leurs effets soient annulés. En mode de mise à jour immédiate, il faut donc défaire dans la BD les mises à jour de ces transactions en réécrivant les anciennes valeurs des données qui devront avoir été mémorisées dans le journal. En mode de mise à jour différée, il n'y a rien à faire pendant la reprise après panne, les modifications des transactions non confirmées n'ayant pas été propagées dans la BD.

On peut maintenant identifier précisément le travail à réaliser aussi bien pour la tenue du journal de reprise que pour la reprise après panne et ce pour chacun des deux modes de mise à jour...

4.6.2 Journalisation et reprise en mode de mise à jour immédiate

Dans ce mode de mise à jour, toute modification est propagée immédiatement dans la BD... via la mémoire cache (donc pas forcément en mémoire de stockage).

4.6.2.1 Déroulement d'une transaction et journalisation

La journalisation en mode de mise à jour immédiate se déroule donc comme suit :

- Pour chaque action `start` d'une transaction T , écrire (T "start") dans le journal,
- Pour chaque écriture d'une donnée D faite par une transaction T :
 - Écrire (T D ancienne_valeur nouvelle_valeur) dans le journal,
 - Puis écrire le dernier bloc du journal dans la mémoire stable,
 - Et enfin écrire D dans la BD (en mémoire cache, donc).
- Pour chaque action `commit` d'une transaction T , écrire (T "commit") dans le journal puis écrire le dernier bloc du journal dans la mémoire stable.



Question

Et si, en cas d'écriture, une panne survient entre la tenue du journal de reprise et l'écriture de la donnée ?

Réponse

Pas de souci : au pire, le journal de reprise nous indiquera de remettre l'ancienne valeur de la donnée dans la BD alors que c'est en fait toujours cette ancienne valeur qui est dans la BD. On va donc juste l'écraser par elle-même...

4.6.2.2 Reprise après panne

Du coup, en mode de mise à jour immédiate, la reprise après panne se fait comme suit :

1. On déroule le journal de reprise à l'envers (i.e. depuis la fin) jusqu'au dernier point de reprise⁴⁶ en notant les transactions confirmées et celles qui ne le sont pas,
2. On déroule de nouveau le journal à l'envers (i.e. depuis la fin) jusqu'au dernier point de reprise⁴⁶ : pour chaque événement (T D a n) d'une transaction non confirmée T , écrire l'ancienne valeur a de D dans la BD (via la mémoire cache, donc),
3. On déroule le journal à l'endroit depuis le dernier point de reprise : pour chaque événement (T D a n) d'une transaction confirmée T , écrire la nouvelle valeur n de D dans la BD (toujours via la mémoire cache, donc).



En pratique

Les deux premières étapes sont souvent réalisées en même temps en une seule passe (donc une seule « remontée est faite dans le journal de reprise»).

⁴⁶ Donc le premier rencontré puisqu'on lit le journal de reprise à l'envers.

Exemple

Soit l'exécution concurrente E suivante, de transactions à deux phases sans opérations de reclassement de verrou, réalisée en mode de mise à jour immédiate :

E		Remarques
	T ₁	T ₂
1	start	1
2	lock X A	2
3	read A in varA ₁	3
4	varA ₁ := varA ₁ + 25	4
5	write A from varA₁	5 <i>T₁ écrit A = 45</i>
6		start 6
7		lock X B 7
8		read B in varB ₂ 8 <i>T₂ lit B = 20</i>
9		lock S C 9
10		read C in varC ₂ 10 <i>T₂ lit C = 20</i>
11	lock S C	11
12	read C in varC ₁	12 <i>T₁ lit C = 20</i>
13		varB ₂ := varB ₂ + varC ₂ 13
14	varD ₁ := varC ₁ * 10	14
15	lock X D	15
16	write D from varD₁	16 <i>T₁ écrit D = 200</i>
17	commit	17
18		write B from varB₂ 18 <i>T₂ écrit B = 40</i>
PANNE		

Figure 66. Exemple d'exécution concurrente à journaliser

La portion du journal de reprise correspondant est la suivante :

Journal de reprise
...
("checkpoint")
(T1 "start")
(T1 A 20 45)
(T2 "start")
(T1 D 20 200)
(T1 "commit")
(T2 B 20 40)

● Panne ●

Figure 67. Exemple de journal de reprise (en mode de mise à jour immédiate)

En cas de panne, la reprise se fait comme suit :

1. La remontée dans le journal montre que, au moment de la panne, la transaction T₁ était confirmée mais pas la transaction T₂,
2. On doit donc faire une nouvelle remontée pour annuler les modifications réalisées par la transaction T₂ : on restaure la valeur 20 pour la donnée B,
3. On refait les modifications faites par la transaction T₁ : on réécrit dans la BD la valeur 45 de la donnée A et la valeur 200 de la donnée D.

4.6.3 Journalisation et reprise en mode de mise à jour différée

Dans ce mode de mise à jour, toute modification est propagée dans la BD uniquement à la confirmation de la transaction l'ayant provoquée, mais toujours *via* la mémoire cache (donc pas forcément en mémoire de stockage).

4.6.3.1 Déroulement d'une transaction et journalisation

La journalisation en mode de mise à jour différée se déroule donc comme suit :

- Pour chaque action *start* d'une transaction T , écrire $(T \text{ "start"})$ dans le journal,
- Pour chaque écriture d'une donnée D faite par une transaction T , écrire $(T \ D \ nouvelle_valeur)$ dans le journal,
- Pour chaque action *commit* d'une transaction T :
 - Écrire $(T \text{ "commit"})$ dans le journal,
 - Écrire le dernier bloc du journal dans la mémoire stable : la transaction est alors confirmée,
 - Écrire dans la BD (en mémoire cache, donc) les données modifiées par la transaction T .

4.6.3.2 Reprise après panne

Du coup, en mode de mise à jour différée, la reprise après panne se fait comme suit :

1. On déroule le journal à l'envers (*i.e.* depuis la fin) jusqu'au dernier point de reprise⁴⁶ en notant les transactions confirmées (et uniquement elles),
2. On déroule le journal à l'endroit à partir du dernier point de reprise : pour chaque événement $(T \ D \ n)$ d'une transaction confirmée T , écrire la valeur n de D dans la BD (*via* la mémoire cache, donc).

Exemple

On considère la même exécution concurrente Ξ (cf. Figure 66) mais en mode de mise à jour différée. La portion du journal de reprise correspondant est la suivante :

Journal de reprise
...
$("checkpoint")$
$(T1 \text{ "start"})$
$(T1 \ A \ 45)$
$(T2 \text{ "start"})$
$(T1 \ D \ 200)$
$(T1 \text{ "commit"})$
$(T2 \ B \ 40)$



Figure 68. Exemple de journal de reprise (en mode de mise à jour différée)

En cas de panne, la reprise se fait comme suit :

1. La remontée dans le journal montre là encore que, au moment de la panne, la transaction T_1 était confirmée mais pas la transaction T_2 ,
2. On refait les modifications faites par la transaction T_1 : on réécrit dans la BD la valeur 45 de la donnée A et la valeur 200 de la donnée D .

4.7 Paramètres de la journalisation

Aucun paramètre spécifique n'est requis (le mode de mise à jour dépend du gestionnaire transactionnel, cf. §3.2).

Stratégies de gestion de données relationnelles

Livre 2 : offrir des performances optimales

5 Introduction à l'optimisation

SOMMAIRE DÉTAILLÉ DU CHAPITRE 5

5.1	Stockage physique et logique (OS) des données	92
5.1.1	Mémoire de stockage au niveau physique : les secteurs.....	92
5.1.2	Mémoire de stockage et mémoire de travail au niveau logique (OS) : les blocs	96
5.1.3	Transferts de données mémoire de stockage ↔ mémoire de travail	100
5.1.4	Mémoire de travail au niveau logique (OS) : les pages.....	102
5.1.5	<i>Quid</i> des performances d'accès (lecture/écriture/manipulation) aux données ?	105
5.1.6	Et le SGBD dans tout ça ?	106
5.2	Traitements d'une requête	108

FIGURES DU CHAPITRE 5

Figure 72.	Organisation d'un disque en plateaux	92
Figure 73.	Têtes de lecture/écriture d'un disque	93
Figure 74.	Organisation en pistes d'un disque.....	94
Figure 75.	Organisation en cylindres d'un disque.....	94
Figure 76.	Organisation « en quartiers » d'un plateau	95
Figure 77.	Organisation en secteurs d'un plateau	95
Figure 78.	Secteurs (physiques) et blocs (logiques, OS).....	97
Figure 79.	Taille « brute » vs taille occupée sur disque d'un fichier (Windows 10 1903)	98
Figure 80.	Transferts de données mémoire de stockage ↔ mémoire de travail.....	100
Figure 81.	Transferts de blocs mémoire de stockage ↔ mémoire de travail	100
Figure 82.	Secteurs (physiques), blocs (logiques, OS) et pages (logiques, OS).....	102
Figure 83.	Accès page par page aux données contenues dans un bloc	103
Figure 84.	Contenu (grossier) d'une BD.....	107
Figure 85.	Organisation des pages de stockage des données d'une BD	107
Figure 86.	Processus BPMN de traitement d'une requête	108
Figure 87.	Dispositifs d'optimisation de l'évaluation de requêtes.....	108

TABLEAUX DU CHAPITRE 5

Tableau 21.	Paramètres de la mémoire de stockage du SGBD (au niveau physique)	96
Tableau 22.	Paramètres de la mémoire de stockage du SGBD (au niveau des blocs logiques)	99
Tableau 23.	Paramètres de la mémoire (de stockage et de travail) du SGBD (au niveau des pages logiques)	104
Tableau 24.	Opérations en mémoire (stockage et travail) pour la gestion des données.....	105

ÉQUATIONS DU CHAPITRE 5

Équation 2.	Lien entre taille d'un bloc et taille d'un secteur	97
Équation 3.	Liens entre tailles de page, bloc et secteur	102

Une petite « discussion » au sujet de ce que l'on entend par « mémoire » dans le domaine des BD/SGBD s'impose... Dans ce monde-là, la notion de « mémoire » est, bien sûr, centrale 😊 On distingue cependant la « mémoire de stockage » d'un côté et la « mémoire de travail » de l'autre :

- *La mémoire de stockage* : comme son nom l'indique, il s'agit de « l'emplacement » de stockage des données de la BD. Cette mémoire doit être stable (*i.e.* pérenne) et on préconise⁴⁷ bien sûr de l'accompagner de mécanismes de sauvegarde et/ou de duplication et/ou de versionnement.



En pratique

Les dispositifs de mémoire de stockage les plus courants sont :

- Les disques durs (encore très majoritairement car présentant encore de loin le meilleur rapport prix/performances/taille),
- Les disques ou cartes flash (SSD notamment, encore de façon minoritaire mais de plus en plus fréquemment),
- Les CD/DVD/BD, (ré)inscriptibles ou non (notamment pour des données archivées qui ne seront *a priori* plus modifiées ou très peu),
- Les bandes magnétiques (extrêmement rares de nos jours mais encore parfois utilisées).

Ces dispositifs sont donc des « mémoires mortes » (ROM) mais tout de même (sauf exception) (ré)inscriptibles.

- *La mémoire de travail* : il s'agit là de « l'emplacement » dans lequel les données sont transférées depuis la mémoire de stockage afin de pouvoir les manipuler. Contrairement à la mémoire de stockage, elle n'est pas forcément stable (elle ne l'est d'ailleurs quasiment jamais).



En pratique

La mémoire vive (RAM) des ordinateurs est bien sûr la mémoire de travail par excellence (car présentant encore de loin le meilleur rapport prix/performances/taille).

En pratique, les données contenues dans une BD sont transférées de la mémoire de stockage vers la mémoire de travail (ou en sens inverse) par le SGBD lors de toute opération les concernant. Ce transfert a lieu le plus souvent lorsque l'on a besoin de manipuler une donnée⁴⁸ :

- *Pour la lire* : la donnée est transférée depuis la mémoire de stockage vers la mémoire de travail,
- *Pour l'écrire* : la donnée est transférée en sens inverse.

Ces transferts (on parlera aussi de « chargements » en lecture et de « déchargements » en écriture) entre la mémoire de stockage et la mémoire de travail sont un des points les plus coûteux (en termes de performances) d'un SGBD. C'est pourquoi la majorité des « mécanismes » abordés dans ce cours visent à limiter la taille des données à transférer ET le nombre de ces transferts.

⁴⁷ Mais cela n'est pas obligatoire.

⁴⁸ De façon très schématique et grossière, dans le cas des BD/SGBD « in memory », le SGBD, dès l'ouverture d'une BD, transfère toutes les données de cette BD de la mémoire de stockage vers la mémoire de travail (le transfert inverse étant réalisé à sa fermeture). Très grossièrement, l'avantage est la vitesse supérieure en lecture/écriture de la mémoire de travail par rapport à celle de la mémoire de stockage, l'inconvénient étant le coût.

Dans ce cadre, nous allons étudier 4 mécanismes d'optimisation :

- **De la place occupée par les relations**, via l'organisation physique des données,
- **Des transferts mémoire de stockage ↔ mémoire de travail**, via la mémoire cache,
- **Des accès aux données pertinentes**, via l'indexation,
- **De l'évaluation des requêtes**, via le moteur d'optimisation.

5.1 Stockage physique et logique (OS) des données

Physiquement, ces données sont donc stockées sur une (ou des) mémoire de stockage, dans un ou plusieurs fichiers qui peuvent être répartis sur un ou plusieurs sites (dans le cas de BD distribuées). Le « format » (*i.e.* l'ensemble des paramétrages réalisés) de stockage choisi doit permettre :

- Une utilisation optimale de la mémoire,
- Un accès rapide,
- Des mises à jour faciles.

On appelle couramment **site** un site géographique de stockage de fichiers. En fait, un site désigne de façon plus formelle un disque logique sur lequel sont stockés tout (pour les BD mono-sites) ou partie (pour les BD multisites) des données de la BD. Un disque logique peut être associé à :

- Une partie d'un disque physique (typiquement, une partition principale ou une partition logique au sein d'une partition étendue),
- Un disque physique entier,
- Un ensemble de disques physiques (on parle alors de **clusters de disques**) et/ou de partitions (principales et/ou logiques).

5.1.1 Mémoire de stockage au niveau physique : les secteurs

Les fichiers sont donc stockés sur des sites, *i.e.* sur des disques (logiques donc *in fine* physiques). Les disques sont en fait des empilements de **plateaux** (cf. Figure 69). Les paramètres d'organisation des données sur ces plateaux dépendent de contraintes soit matérielles soit logicielles (dépendantes du contrôleur du disque, de sa densité d'information, ...).

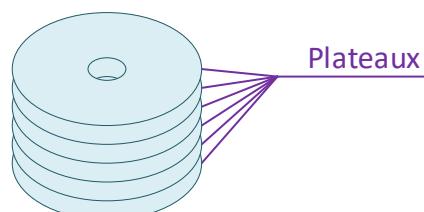


Figure 69. Organisation d'un disque en plateaux



Remarque

Le nombre de plateaux d'un disque dur est un paramètre physique de ce disque : bien qu'il soit possible de « faire croire » à un OS que ce nombre est plus petit⁴⁹ que sa valeur réelle (à l'aide de certains utilitaires « de bas niveau », voire directement dans certains BIOS/EFI), il n'est pas possible de modifier le nombre réel de plateaux d'un disque dur (ou alors il faut changer de disque dur !).

⁴⁹ Jamais plus grand, forcément !

Les têtes de lecture/écriture d'un disque dur (une par plateau) sont « liées » entre-elles : elles sont en réalité toutes positionnées sur un axe de rotation (cf. Figure 70) unique. Ainsi, elles peuvent « bouger » sur cet axe de droite à gauche mais restent toutes solidaires (*i.e.* si une tête tourne à gauche de 15°, toutes les têtes tournent en fait à gauche de 15°). Bien que « solidaires » au niveau de leurs mouvements de rotation, une seule tête de lecture/écriture peut travailler à un moment donné.



En pratique

C'est donc la combinaison de la rotation des plateaux, de l'ensemble des têtes de lecture/écriture et de leur rotation qui rend possible la lecture/l'enregistrement depuis/vers toutes les parties de tous les plateaux du disque dur.

Donc, si une tête « pointe » sur une partie *X* de « son » plateau, toutes les autres têtes « pointent » vers le même endroit *X* de leur plateau respectif mais une seule peut lire ou écrire à la fois.

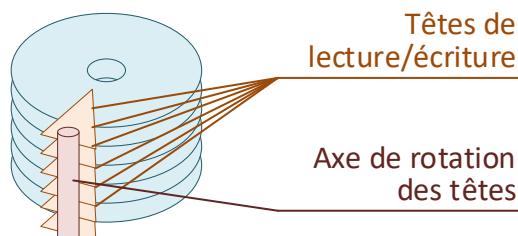


Figure 70. Têtes de lecture/écriture d'un disque



Remarque

Dans certains disques durs, certaines têtes peuvent être « doubles » : elles peuvent alors lire/écrire le plateau qui est au-dessus d'elles mais aussi celui qui est en-dessous (mais PAS simultanément). Le nombre de têtes de lecture/écriture est *in fine* toujours égal au nombre de plateaux du disque dur⁵⁰ mais il arrive donc qu'elles soient « associées » par paires.



En pratique

Ainsi, maximiser les performances d'accès aux données débute au niveau des plateaux des disques durs sur lesquels elles sont stockées : plus ils tournent vite sur eux-mêmes, plus vite les données pourront être accédées et lues (on rencontre des vitesses de rotation de l'ordre de 5 400 rpm⁵¹, 7 200 rpm ou encore 10 000 rpm le plus couramment). Cependant, la vitesse de rotation des plateaux ne peut pas être paramétrée. La vitesse de rotation des têtes de lecture/écriture est elle-aussi cruciale : là non plus, elle ne peut pas être paramétrée. Minimiser les temps de lecture/écriture sur le disque suppose donc de faire bouger le moins possible les têtes au fur et à mesure de ces opérations. Cependant, cette optimisation est gérée par le contrôleur du disque (voire en partie par le système d'exploitation). Ainsi, la rotation des plateaux et celle des têtes de lecture/écriture reposent toutes deux « sur de la mécanique » (par définition moins performante que ce qui est purement électronique). C'est donc un « goulet d'étranglement » classique pour l'accès aux données. D'où l'intérêt de bien choisir son disque de stockage !

⁵⁰ D'ailleurs, dans les caractéristiques des disques durs, c'est quasiment toujours le nombre de têtes de lecture/écriture (*heads*) qui est indiqué (aussi bien sur le disque lui-même que dans les BIOS/EFI).

⁵¹ Tour par minute (*rotation per minute*).

Sur chaque plateau, les données sont organisées en cercles concentriques appelés **pistes**⁵² (cf. Figure 71). Ces pistes sont créées par le formatage de bas niveau. Les têtes commencent à inscrire des données à la périphérie du disque (piste 0), puis avancent vers le centre.

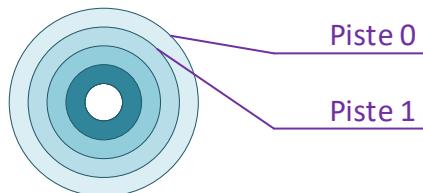


Figure 71. Organisation en pistes d'un disque



Remarque

Le nombre de pistes par plateau d'un disque dur est un paramètre physique de ce disque : bien qu'il soit possible de « faire croire » à un OS que ce nombre est plus petit⁵³ que sa valeur réelle (à l'aide de certains utilitaires « de bas niveau », voire directement dans certains BIOS/EFI), il n'est pas possible de modifier le nombre réel de pistes par plateau d'un disque dur (ou alors il faut changer de disque dur !).

Ce nombre de pistes par plateau dépend principalement de la densité informationnelle des plateaux et de la précision des têtes de lecture/écriture : plus cette densité est élevée, plus on peut mettre de pistes par plateau et plus on pourra stocker de données sur chaque plateau.

On parle de **cylindre** pour désigner l'ensemble des données situées sur une même piste des plateaux (c'est-à-dire à la verticale les unes des autres) : cela forme dans l'espace un « cylindre » de données (cf. Figure 72). Le nombre de cylindres est bien sûr égal au nombre de pistes par plateau du disque dur.

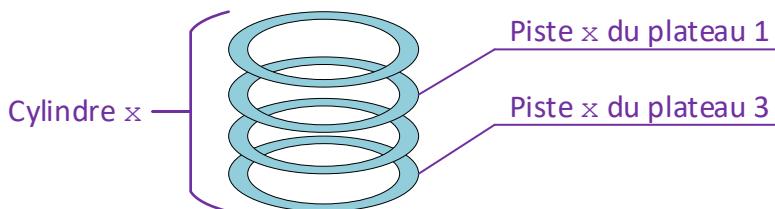


Figure 72. Organisation en cylindres d'un disque



En pratique

Le nombre de cylindres est donc égal au nombre de pistes par plateau⁵⁴. *A priori*, plus ce nombre est élevé et plus la densité informationnelle des plateaux du disque est bonne, ce qui est signe de pouvoir stocker une bonne quantité de données et/ou d'avoir de bonnes performances d'accès à ces données (mais, comme indiqué plus haut, cela dépend aussi de la vitesse de rotation des plateaux et des caractéristiques des têtes de lecture/écriture, entre autres facteurs).

⁵² Et non en un sillon (comme sur les anciens disques analogiques mais aussi sur les CD/DVD/BD) comme on le croit encore trop souvent à tort 😞

⁵³ Jamais plus grand, forcément !

⁵⁴ D'ailleurs, dans les caractéristiques des disques durs, c'est quasiment toujours le nombre de cylindres (*cylinders*) qui est indiqué (aussi bien sur le disque lui-même que dans les BIOS/EFI).

Les plateaux sont aussi divisés « en quartiers » (cf. Figure 73).

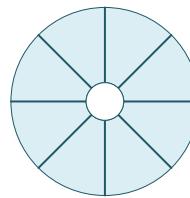


Figure 73. Organisation « en quartiers » d'un plateau

L'intersection de ces quartiers et des pistes définit un ensemble de **secteurs** (cf. Figure 74). Le nombre de secteurs dépendant du nombre de pistes par plateau et du nombre de quartiers par plateau, c'est un paramètre physique du disque. **Chaque secteur est identifiable de façon unique sur « sa » mémoire de stockage par son identificateur $id_{secteur}$.**



Attention

L'identificateur $id_{secteur}$ d'un secteur n'a AUCUNE raison d'être stocké dans ledit secteur (de la même façon que, en programmation, l'indice d'une case d'un tableau n'est PAS stocké dans cette case) : ces identificateurs de secteurs $id_{secteur}$ sont en réalité gérés « ailleurs » (dans le contrôleur disque et/ou par l'OS).

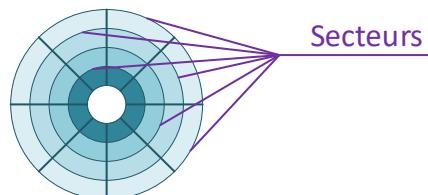


Figure 74. Organisation en secteurs d'un plateau



Remarque

Il est à noter que, bien que les secteurs d'un même disque n'aient pas tous la même taille physique, ils ont tout de même tous la même capacité de stockage (cela simplifie notamment la gestion du stockage des données sur le disque dur, donc le contrôleur du disque dur).



En pratique

Plus la taille d'un secteur est petite, moins on aura de risque de perdre de la place sur le disque. Mais, avoir des secteurs de taille plus petite implique d'avoir à gérer plus de secteurs, donc à complexifier notamment le contrôleur disque...



Définition : « secteur »

Le **secteur** est l'unité minimale physique de stockage sur un disque (on parle « d'unité physique d'allocation »). Sa capacité dépend de la densité de stockage du disque et du paramétrage réalisé en usine par le fabricant de ce disque.

Un fichier peut occuper plusieurs secteurs mais un secteur ne peut pas contenir de données issues de plusieurs fichiers ! Ceci cause des « pertes » en capacité de stockage...



Exemple

Supposons qu'on stocke des données sur un disque dur dont les secteurs font 512 octets. Si on stocke un fichier occupant en réalité 68 octets, il occupe effectivement 1 secteur entier (444 octets de ce secteur étant alors « perdus »). De même, si on stocke un fichier occupant en réalité 1 267 octets, il occupe effectivement 3 secteurs entiers (2 sont pleins et le dernier contient les 243 octets restants, les 269 octets inoccupés de ce dernier secteur étant alors « perdus »).

En théorie⁵⁵, les paramètres liés à la gestion physique (*i.e.* aux secteurs) des mémoires de stockage et susceptibles de nous intéresser au niveau des mesures de performances des SGBD sont les suivants :

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$tps_{lectSecteur}^{fixe}(MS)$	ms	Temps de lecture (chargement) d'un secteur depuis la mémoire de stockage MS (considéré fixe pour une mémoire de stockage donnée).
$tps_{ecritSecteur}^{fixe}(MS)$	ms	Temps d'écriture (déchargement) d'un secteur vers la mémoire de stockage MS (considéré fixe pour une mémoire de stockage donnée).
$T_{secteur}^{fixe}(MS)$	octets	Taille d'un secteur de la mémoire de stockage MS (considérée fixe pour une mémoire de stockage donnée).
$nb_{secteurs}^{fixe}(MS)$	secteurs	Nombre (fixe) de secteurs utilisables sur la mémoire de stockage MS (au total, <i>i.e.</i> qu'ils soient « libres » ou déjà « occupés »).
$T_{idSecteur}^{fixe}(MS)$	octets	Taille d'un identificateur de secteur sur la mémoire de stockage MS (considérée fixe pour une mémoire de stockage donnée).

Tableau 21. Paramètres de la mémoire de stockage du SGBD (au niveau physique)

5.1.2 Mémoire de stockage et mémoire de travail au niveau logique (OS) : les blocs

Les systèmes d'exploitation (OS) définissent lors du formatage du disque (formatage de haut niveau) leur propre unité minimale de stockage (« unité logique d'allocation », cf. Figure 75) :

- Cette unité minimale logique de stockage est appelée **cluster** ou **bloc**,
- Un cluster a toujours une capacité qui est un multiple (une puissance de 2) de la capacité de stockage des secteurs du disque sur lequel il se trouve.



Question

Pourquoi les OS définissent leur propre unité minimale de stockage (logique) ?

Réponse

Pour simplifier la gestion du stockage des données à leur niveau, tout simplement... Ce mécanisme a été notamment « imposé » lorsque la taille des disques durs a commencé à « exploser ».

⁵⁵ Ces paramètres sont rarement indiqués (c'est pourquoi ils sont écrits en gris dans le tableau) : en raison des hypothèses simplificatrices que nous adopterons, ils seront en effet rarement utiles (si c'est le cas, ce sera pour retrouver un autre paramètre qui ne nous aurait pas été directement fourni).



Définition : « bloc »

Le **bloc** est l'unité minimale logique de stockage au niveau du système d'exploitation (on parle « d'unité logique d'allocation du système d'exploitation »). La taille d'un bloc s'exprime toujours de la façon suivante⁵⁶ :

$$T_{bloc}^{fixe} = 2^{i(MS,OS)} \times T_{secteur}^{fixe}$$

(avec $i(MS, OS)$ un entier tel que $i(MS, OS) \geq 0$)

Équation 2. Lien entre taille d'un bloc et taille d'un secteur

Un fichier peut occuper en mémoire de stockage plusieurs blocs mais un bloc ne peut pas contenir de données issues de plusieurs fichiers ! Ceci cause des « pertes » en capacité de stockage...



En pratique

La notion de bloc permet à l'OS de faire le lien entre la gestion des données en mémoire de stockage (*via* les secteurs, cf. §5.1.1) et la gestion des données en mémoire de travail (*via* les pages, cf. §5.1.4).

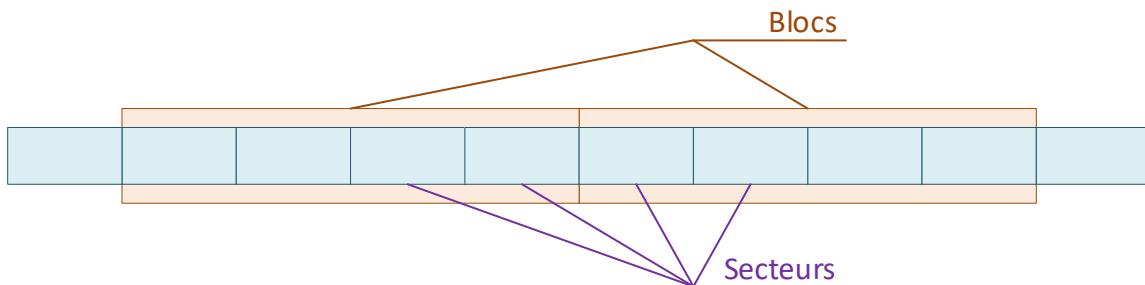


Figure 75. Secteurs (physiques) et blocs (logiques, OS)

Ce sont ces clusters (ou blocs) qui sont exploités par le système d'exploitation pour stocker les fichiers. Chaque bloc est identifiable de façon unique sur « sa » mémoire de stockage par son identificateur id_{bloc} .



Attention

L'identificateur id_{bloc} d'un bloc n'a AUCUNE raison d'être stocké dans ledit bloc (de la même façon que, en programmation, l'indice d'une case d'un tableau n'est PAS stocké dans cette case) : ces identificateurs de blocs id_{bloc} sont en réalité gérés « ailleurs » (par l'OS).



En pratique

Tout comme pour les secteurs, plus la taille d'un bloc est petite, moins on aura de risque de perdre de la place sur le disque. Cependant, là aussi, plus la taille des blocs sera petite et plus leur nombre sera grand, donc plus il sera « complexe » de les gérer.

⁵⁶ La valeur de i est propre au couple mémoire de stockage/système d'exploitation.

Ainsi, un fichier minuscule (dont la taille est inférieure à celle d'un bloc) devra donc tout de même occuper un bloc complet au minimum (et, donc, des secteurs pourront être « perdus », *i.e.* utilisés inutilement). Les fichiers stockés sur disque s'étendent donc :

- Au niveau logique, sur un ou plusieurs blocs,
- Au niveau physique, sur un nombre de secteurs étant un multiple (forcément une puissance de 2) du nombre de blocs occupés au niveau logique.



Exemple

Supposons qu'on stocke des données sur un disque dur dont les secteurs font 512 octets et que ce disque est géré par un OS dont les blocs font 1 024 octets (1 bloc = 2^1 secteurs). Si on stocke un fichier occupant en réalité 68 octets, il occupe effectivement 1 bloc entier : le 1^{er} secteur de ce bloc est partiellement occupé (68 octets utilisés et 444 octets « perdus ») et le 2nd secteur de ce bloc est entièrement « perdu » (ses 512 octets sont « inutilisés »). De même, si on stocke un fichier occupant en réalité 1 267 octets, il occupe effectivement 2 blocs : les 2 secteurs du 1^{er} bloc sont entièrement occupés alors que le 1^{er} secteur du 2nd bloc l'est partiellement (à 243 octets, les 269 octets restants de ce secteur étant « inutilisés ») et le 2nd secteur du 2nd bloc est, lui, totalement « perdu » (ses 512 octets sont « inutilisés »).

Les OS modernes montrent souvent bien la différence entre taille des données (réelle) et taille de ces données en mémoire de stockage (la différence étant donc notamment due à la place « perdue » à cause des mécanismes de blocs et de secteurs).

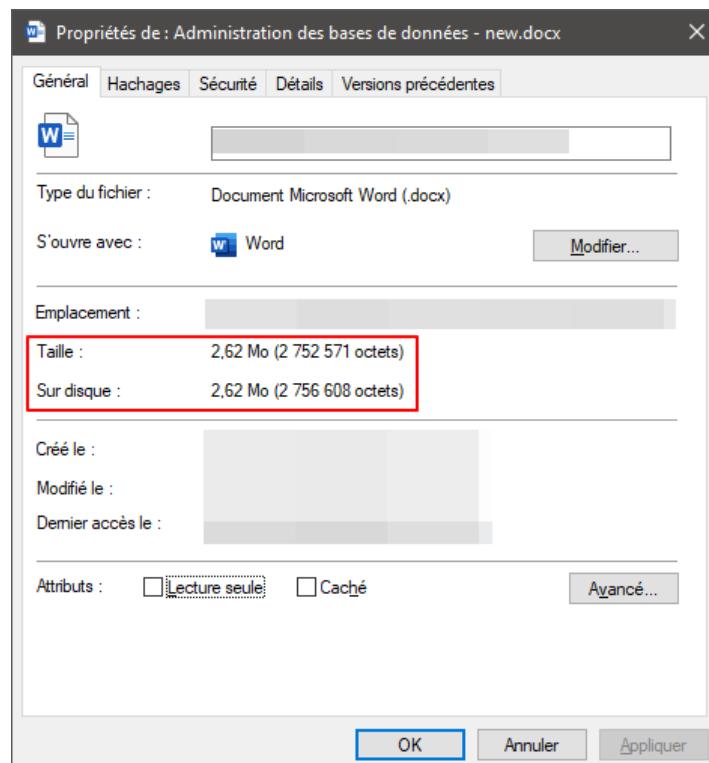


Figure 76. Taille « brute » vs taille occupée sur disque d'un fichier (Windows 10 1903)

Un fichier est identifié par un nom logique et est constitué d'un ensemble de blocs, pas forcément contigus sur le disque dur, au niveau logique (et donc de secteurs au niveau physique). L'espace occupé par un fichier sur un disque dur varie au cours du temps. On distingue quatre phases : l'allocation initiale, l'expansion, la contraction et la réorganisation).

En théorie⁵⁷, les paramètres liés à la gestion logique des blocs des mémoires de stockage et susceptibles de nous intéresser au niveau des mesures de performances des SGBD sont les suivants :

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$tps_{lectBloc}^{fixe}(Mem, OS)$	ms	Temps de lecture (chargement) d'un bloc (considéré fixe pour une mémoire, de stockage ou de travail, donnée et un OS donné).
$tps_{ecritBloc}^{fixe}(Mem, OS)$	ms	Temps d'écriture (déchargement) d'un bloc (considéré fixe pour une mémoire, de stockage ou de travail, donnée et un OS donné).
$T_{bloc}^{fixe}(Mem, OS)$	octets	Taille d'un bloc (considérée fixe pour une mémoire, de stockage ou de travail, donnée et un OS donné) ⁵⁸ : $T_{bloc}^{fixe}(MS, OS) = 2^{i(MS, OS)} \times T_{secteur}^{fixe}(MS)$ <p>(avec $i(MS, OS)$ un entier positif ou nul)</p>
$nb_{secteurs/bloc}^{fixe}(MS, OS)$	secteurs par bloc	Nombre (fixe pour une mémoire, <u>forcément de stockage</u> , et un OS donné) de secteurs occupés par un bloc : $nb_{secteurs/bloc}^{fixe}(MS, OS) = \frac{T_{bloc}^{fixe}(MS, OS)}{T_{secteur}^{fixe}(MS)}$
$nb_{blocs}^{fixe}(Mem, OS)$	blocs	Nombre (fixe) de blocs utilisables sur une mémoire, de stockage ou de travail, donnée via l'OS (au total, i.e. qu'ils soient « libres » ou déjà « occupés ») : $nb_{blocs}^{fixe}(MS, OS) = nb_{secteurs}^{fixe}(MS) \times nb_{secteurs/bloc}^{fixe}(MS, OS)$
$T_{idBloc}^{fixe}(OS)$	octets	Taille d'un identificateur de bloc (considérée fixe pour un OS donné).

Tableau 22. Paramètres de la mémoire de stockage du SGBD (au niveau des blocs logiques)

⁵⁷ Ces paramètres, sont rarement indiqués (c'est pourquoi ils sont écrits en gris dans le tableau) : en raison des hypothèses simplificatrices que nous adopterons, ils seront en effet rarement utiles (si c'est le cas, ce sera pour retrouver un autre paramètre qui ne nous aurait pas été directement fourni).

⁵⁸ La valeur de i est propre au couple mémoire de stockage/système d'exploitation.

5.1.3 Transferts de données mémoire de stockage ↔ mémoire de travail

Les logiciels applicatifs ne manipulent jamais directement les données en mémoire de stockage : une des « missions » de l'OS est :

- De transférer les données adéquates de la mémoire de stockage vers la mémoire de travail afin de les rendre accessibles aux logiciels applicatifs,
- De transférer les données modifiées en mémoire de travail par les logiciels applicatifs vers la mémoire de stockage (afin de rendre ces modifications pérennes).

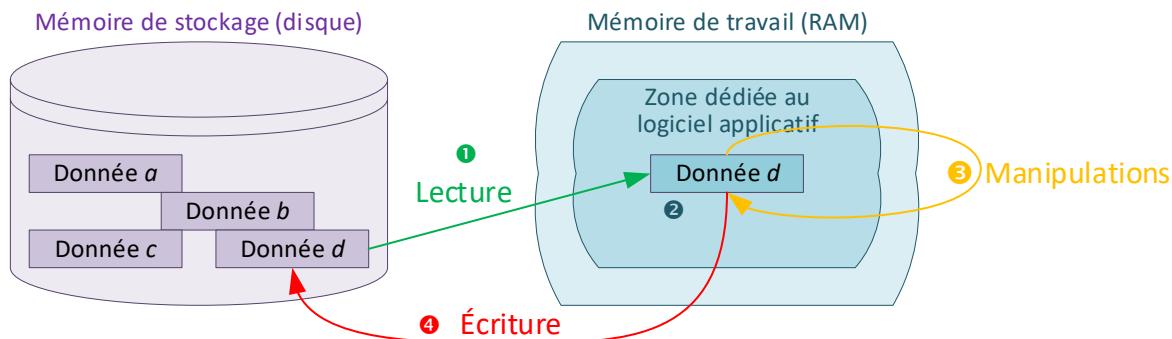


Figure 77. Transferts de données mémoire de stockage ↔ mémoire de travail

De façon grossière, lorsqu'un logiciel applicatif a besoin de travailler sur une donnée *d* (cf. Figure 77) :

1. Le logiciel applicatif demande à l'OS d'accéder à cette donnée *d*. Si l'OS « valide » cette demande, il transfère ladite donnée depuis la mémoire de stockage vers la zone de la mémoire de travail qu'il a dédiée au logiciel applicatif demandeur : c'est la lecture de la donnée.
2. La donnée *d* est maintenant disponible en mémoire de travail pour que le logiciel applicatif puisse y accéder.
3. Le logiciel applicatif réalise toutes les manipulations nécessaires sur la donnée *d*.
4. Si le logiciel applicatif a besoin de pérenniser des modifications faites sur la donnée *d*, il demande à l'OS de la sauvegarder. Si l'OS « valide » cette demande, il transfère ladite donnée depuis la zone de la mémoire de travail qu'il a dédiée au logiciel applicatif demandeur vers la mémoire de stockage : c'est l'écriture de la donnée.

Étant donné que l'OS considère les données enregistrées en mémoire de stockage par blocs (cf. §5.1.2), les transferts mémoire de stockage ↔ mémoire de travail se font en réalité par blocs (cf. Figure 78).

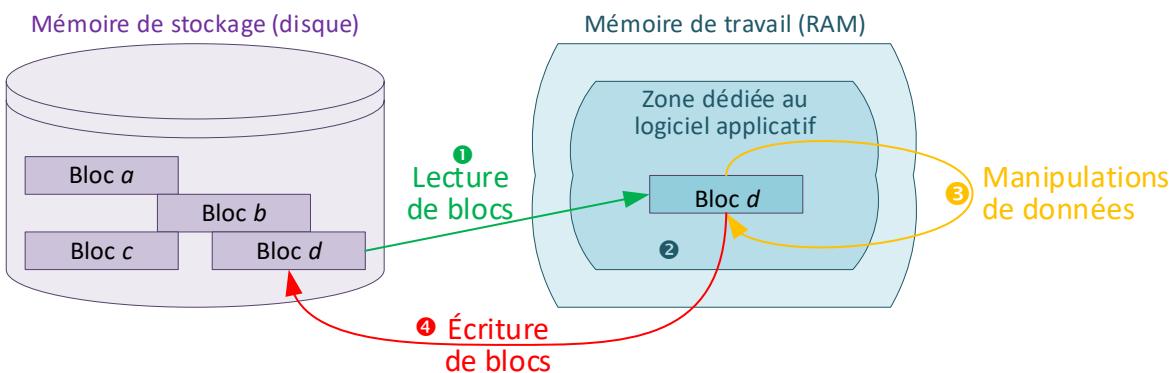


Figure 78. Transferts de blocs mémoire de stockage ↔ mémoire de travail

Ainsi, en précisant avec cette notion de blocs les accès aux données :

1. Le logiciel applicatif demande à l'OS d'accéder à cette donnée *d*. Si l'OS « valide » cette demande, il transfère le ou les blocs qui contiennent ladite donnée depuis la mémoire de stockage vers la zone de la mémoire de travail qu'il a dédiée au logiciel applicatif demandeur : c'est la lecture de la donnée (en fait du ou des blocs qui la contiennent).
2. La donnée *d* est maintenant disponible en mémoire de travail pour que le logiciel applicatif puisse y accéder.
3. Le logiciel applicatif réalise toutes les manipulations nécessaires sur la donnée *d*.
4. Si le logiciel applicatif a besoin de pérenniser des modifications faites sur la donnée *d*, il demande à l'OS de la sauvegarder. Si l'OS « valide » cette demande, il transfère le ou les blocs qui contiennent ladite donnée depuis la zone de la mémoire de travail qu'il a dédiée au logiciel applicatif demandeur vers la mémoire de stockage : c'est l'écriture de la donnée (en fait du ou des blocs qui la contiennent).



Définition : « transfert »

Il y a **transfert** (de blocs) lors de la copie de données (en fait des blocs qui les contiennent) depuis la mémoire de stockage vers la mémoire de travail (pour les rendre accessibles) ou dans l'autre sens (pour les sauvegarder).



Définition : « chargement » ou « transfert en lecture »

Il y a **chargement** (de blocs) lors de la copie de données (en fait des blocs qui les contiennent) depuis la mémoire de stockage vers la mémoire de travail, donc dans le sens de la lecture. Un chargement est donc un transfert en lecture.



Définition : « déchargement » ou « transfert en écriture »

Il y a **déchargement** (de blocs) lors de la copie de données (en fait des blocs qui les contiennent) depuis la mémoire de travail vers la mémoire de stockage, donc dans le sens de l'écriture. Un déchargement est donc un transfert en écriture.

Ainsi, on se trouve ici face à plusieurs types de **transferts de données (par blocs)** (cf. 5.1.2) :

- *De la mémoire de stockage vers la mémoire de stockage (cas exceptionnel)* : par exemple en cas de défragmentation des données stockées sur une mémoire de stockage⁵⁹,
- *De la mémoire de stockage vers la mémoire de travail (courant)* : pour lire les données stockées, ce sont les **chargements**,
- *De la mémoire de travail vers la mémoire de stockage (« rare »)* : pour stocker des nouvelles données ou des modifications apportées à des données existantes, ce sont les **déchargements**,
- *De la mémoire de travail vers la mémoire de travail (très fréquent)* : la RAM étant découpée en plusieurs zones, il est très fréquent d'avoir à transférer des données depuis une zone de la mémoire de travail vers une autre zone de la mémoire de travail.

⁵⁹ Et encore : il s'agit plus en réalité de transferts de la mémoire de stockage vers la mémoire de travail puis de transferts inverse mais à un autre emplacement de la mémoire de stockage.

5.1.4 Mémoire de travail au niveau logique (OS) : les pages

Une fois transférée depuis la mémoire de stockage vers la mémoire de travail (dans la zone dédiée à l'application demanduse), les données sont disponibles pour être manipulées. Afin de faciliter cette manipulation, l'OS adopte un « nouveau découpage » de la mémoire de travail : la **page**⁶⁰.



Définition : « page »

Une page est l'unité minimale de stockage logique en mémoire de travail au niveau de l'OS. La taille d'une page s'exprime toujours de la façon suivante⁶¹ :

$$T_{page}^{fixe}(OS) = 2^{i(OS, MS)} \times T_{secteur}^{fixe}(MS)$$

$$\text{et } T_{bloc}^{fixe}(MS, OS) = 2^{j(OS)} \times T_{page}^{fixe}(OS)$$

$$(avec T_{secteur}^{fixe}(MS) \leq T_{page}^{fixe}(OS) \leq T_{bloc}^{fixe}(Mem, OS),$$

$$i(OS, MS) \text{ et } j(OS) \text{ des entiers tels que } i(OS, MS) \geq 0 \text{ et } j(OS) \geq 0)$$

Équation 3. Liens entre tailles de page, bloc et secteur

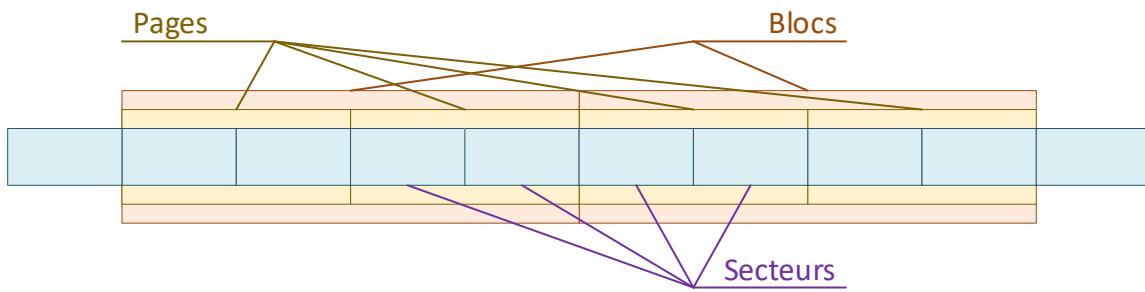


Figure 79. Secteurs (physiques), blocs (logiques, OS) et pages (logiques, OS)



Question

Pourquoi définir 2 unités minimales : les blocs au niveau de la mémoire de stockage et les pages au niveau de la mémoire de travail ?

Réponse

L'idéal serait de pouvoir accéder aux données octet par octet. Cependant, la taille des mémoires de stockage est devenue tellement grande que cela deviendrait trop complexe. D'où les notions de secteurs (au niveau physique de la mémoire de stockage) et de blocs (au niveau logique de la mémoire de stockage). En revanche, la taille des mémoires de travail reste nettement plus faible : elle peut donc être gérée plus finement sans accroître la complexité des accès. D'où la notion de pages (au niveau logique de la mémoire de travail).

⁶⁰ C'est le concept de mémoire paginée, démocratisé vers 1986 par l'arrivée du processeur Intel 80386 (source : [https://fr.wikipedia.org/wiki/M%C3%A9moire_pagin%C3%A9e_\(MS-DOS\)](https://fr.wikipedia.org/wiki/M%C3%A9moire_pagin%C3%A9e_(MS-DOS))).

⁶¹ La valeur de i est propre au couple mémoire de stockage/système d'exploitation ; celle de j est propre à l'OS.

Une page est l'unité d'allocation minimale (logique) d'un OS en mémoire de travail. La capacité d'une page est inférieure ou égale à celle d'un bloc : cette capacité est un multiple (en réalité une puissance de 2) de la capacité d'un secteur. Une page peut donc être vue comme une partie d'un bloc et, donc, un bloc contient usuellement plusieurs pages (cf. Figure 79). **Chaque page est identifiable de façon unique au sein de « son » bloc par son identificateur id_{page} .**



Attention

L'identificateur id_{page} d'une page n'a AUCUNE raison d'être stocké dans ladite page (de la même façon que, en programmation, l'indice d'une case d'un tableau n'est PAS stocké dans cette case) : ces identificateurs de pages id_{page} sont en réalité générés « ailleurs » (par l'OS).



Attention

Les pages sont donc la plus petite quantité de données logique adressable par l'OS (et, donc, par les logiciels applicatifs, dont les SGBD) en mémoire de travail. Ainsi :

- *En mémoire de travail* : les données sont adressées page par page par l'OS,
- *En mémoire de stockage* : les données sont adressées bloc par bloc par l'OS ; mais, un bloc pouvant être vu comme un ensemble de pages, on conserve une certaine homogénéité entre les 2 types de mémoires, bien que fondamentalement adressées différemment.

Ainsi, au niveau logique, on peut « considérer » que tout est stocké au sein de pages (elles-mêmes ensuite groupées par bloc).

Ainsi, accéder aux données contenues dans un bloc en mémoire de travail ne nécessite pas de travailler sur tout le bloc : il suffit d'accéder à la page adéquate au sein dudit bloc.

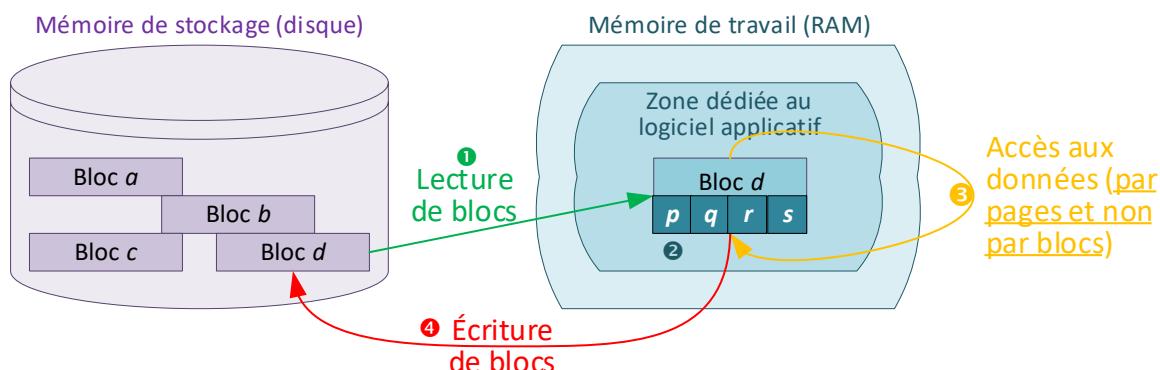


Figure 80. Accès page par page aux données contenues dans un bloc

En théorie⁶², les paramètres liés à la gestion logique des pages de la mémoire de travail⁶³ et susceptibles de nous intéresser au niveau des mesures de performances des SGBD sont les suivants :

⁶² Ces paramètres, sont rarement tous indiqués (c'est le cas de ceux qui sont écrits en gris dans le tableau ci-après) : en raison des hypothèses simplificatrices que nous adopterons, ils seront en effet rarement tous utiles (si c'est le cas, ce sera pour retrouver un autre paramètre qui ne nous aurait pas été directement fourni).

⁶³ Et, « par ricochet », de la mémoire de stockage.

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$tps_{lectPage}^{fixe}(Mem, OS)$	ms	Temps de lecture d'une page depuis la mémoire, de travail ou de stockage (considéré fixe pour une mémoire et un OS donnés). Négligeable pour une mémoire de travail !
$tps_{ecritPage}^{fixe}(Mem, OS)$	ms	Temps d'écriture d'une page depuis la mémoire, de travail ou de stockage (considéré fixe pour une mémoire et un OS donnés). Négligeable pour une mémoire de travail !
$T_{page}^{fixe}(OS)$	octets	<p>Taille d'une page (considérée fixe pour un OS donné) :</p> $T_{page}^{fixe}(OS) = 2^{i(MS, OS)} \times T_{secteur}^{fixe}(MS)$ <p>et $T_{bloc}^{fixe}(MS, OS) = 2^{j(OS)} \times T_{page}^{fixe}(OS)$</p> <p>(avec $T_{secteur}^{fixe}(MS) \leq T_{page}^{fixe}(OS) \leq T_{bloc}^{fixe}(MS, OS)$, $i(MS, OS)$ et $j(OS)$ des entiers positifs ou nuls)</p>
$nb_{pages}^{fixe}(Mem, OS)$	pages	Nombre (fixe) de pages utilisables sur la mémoire (stockage ou travail) par l'OS (« libres » ou « occupés »).
$nb_{secteurs/page}^{fixe}(MS, OS)$	secteurs par page	<p>Nombre (fixe pour une mémoire <u>de stockage</u> et un OS donné) de secteurs « occupés » par une page :</p> $nb_{secteurs/page}^{fixe}(MS, OS) = \frac{T_{page}^{fixe}(MS, OS)}{T_{secteur}^{fixe}(MS)}$
$nb_{pages/bloc}^{fixe}(OS)$	pages par bloc	<p>Nombre (fixe pour un OS donné) de pages au sein d'un bloc :</p> $nb_{pages/bloc}^{fixe}(MS, OS) = \frac{T_{bloc}^{fixe}(MS, OS)}{T_{page}^{fixe}(OS)}$
$nb_{pages}^{fixe}(Mem, OS)$	blocs	<p>Nombre (fixe) de pages utilisables sur la mémoire (stockage ou travail) via l'OS (« libres » ou « occupées ») :</p> $nb_{pages}^{fixe}(Mem, OS) = nb_{blocs}^{fixe}(Mem) \times nb_{pages/bloc}^{fixe}(OS)$
$T_{idPage}^{fixe}(OS)$	octets	Taille d'un identificateur de page (considérée fixe pour un OS donné).

Tableau 23. Paramètres de la mémoire (de stockage et de travail) du SGBD (au niveau des pages logiques)

5.1.5 **Quid des performances d'accès (lecture/écriture/manipulation) aux données ?**

Les SGBD sont, par définition, gros consommateurs de ressources étant donné qu'ils réalisent énormément d'accès aux données (pour les lire, les écrire et les manipuler). Ils doivent néanmoins conserver des performances raisonnables (le plus possible) !

Au niveau de la gestion des données, on considère finalement 5 opérations :

Opération	Performance (temps)
Manipulation en mémoire de travail	Négligeable (quelques ns)
Transfert	Mémoire de stockage → mémoire de stockage Non traité (« inutile » dans le cadre des SGBD)
	Mémoire de stockage → mémoire de travail Transfert de blocs en lecture (chargement), quelques ms
	Mémoire de travail → mémoire de stockage Transfert de blocs en écriture (déchargement), quelques ms
	Mémoire de travail → mémoire de travail Négligeable (quelques ns)

Tableau 24. Opérations en mémoire (stockage et travail) pour la gestion des données

On le voit, ce qui est coûteux ce sont surtout les transferts entre mémoire de stockage et mémoire de travail (en lecture aussi bien qu'en écriture). **Les 3 autres opérations auront un coût considéré dans ce cours comme négligeable !**

Cette hypothèse étant posée, **maximiser les performances d'un SGBD consiste donc, entre autres choses, à minimiser le nombre de ces transferts (chargements et déchargements).** Nous allons donc, discuter de la minimisation du nombre de ces transferts. Mais cela n'est pas tout⁶⁴ : offrir de bonnes performances à un SGBD implique, entre autres aspects, de trouver un bon équilibre entre :

- *La place occupée* : elle est à minimiser, de sorte à occuper le moins de place possible sur la mémoire de stockage et, donc, à avoir le moins possible de transferts à réaliser (chargements et déchargements),
- *La facilité de gestion (des données, des pages, ...)* par le SGBD : plus cette facilité sera grande, moins les opérations de gestion seront-elles-mêmes coûteuses en performances ; selon les cas, les stratégies choisies via le paramétrage du SGBD demanderont plus ou moins de transferts.



Remarque

Notez qu'on ne cherche en fait jamais un « optimum », ni du côté de la place occupée, ni du côté de la facilité de gestion : tout sera toujours une question de bon compromis, d'où le fait que plusieurs stratégies soient régulièrement proposées, chacune étant « usuellement » plus adaptée à un certain nombre de cas mais pas forcément à d'autres (d'où aussi la nécessité d'avoir de bons administrateurs de bases de données ! 😊).

⁶⁴ C'est en effet nécessaire mais pas suffisant.



En pratique

On peut ainsi estimer la performance d'un traitement réalisé par un SGBD en évaluant le nombre de transferts qu'il effectue pour réaliser ledit traitement : plus ce nombre est petit, meilleure est la performance de réalisation de ce traitement. Cependant, les mémoires de stockage étant ce qu'elles sont, il est fréquent que le temps de chargement ne soit pas égal au temps de déchargement (les temps d'accès en lecture sont souvent inférieurs (rarement égaux) aux temps d'accès en écriture). Il est donc usuel d'estimer la performance en temps. Dans ce cas, on compte le nombre de chargements (ce qui permet de connaître le temps global utilisé pour réaliser ces opérations de lecture) ET le nombre de déchargement (ce qui permet de connaître le temps global utilisé pour réaliser ces opérations d'écriture) : la somme des 2 temps obtenus donne une évaluation du temps nécessaire pour réaliser le traitement. **Dans tous les cas, chacun de ces 2 temps (temps de lecture et temps d'écriture) est considéré uniforme pour un disque physique donné⁶⁵.** Ainsi, pour un disque donné, le temps d'accès en lecture (chargement) est supposé toujours le même. Il en va de même pour le temps d'accès en écriture. Par exemple, on peut vous indiquer pour un disque donné $t_L = 1\text{ms}$ et $t_E = 1,5\text{ms}$, ce qui signifie que toute opération de chargement depuis ce disque (bloc/page) prend 1ms et toute opération de déchargement vers ce disque (bloc/page) prend 1,5ms.

5.1.6 Et le SGBD dans tout ça ?

On l'a dit, un SGBD manipule toute une foule de données, organisées en BD, contenant notamment :

- Des relations, basées sur un modèle (intension, cf. §1.3.1.2.1) et constituées de n-uplets (extension, cf. §1.3.1.2.1), eux-mêmes étant une collection de valeurs d'attributs (ceux indiqués dans le modèle),
- Des index, contenant principalement des entrées d'index et des informations de circulation dans l'index (cf. §8).



En pratique

En réalité, un SGBD manipule bien d'autres choses encore, mais nous nous limiterons dans ce cours aux relations et aux index comme constituants des BD.

Comme pour tout logiciel, les données manipulées par un SGBD sont organisées en pages (cf. §5.1.4), notamment ici (cf. Figure 81) les n-uplets⁶⁶ (*i.e.* les lignes des tables représentant les relations) mais aussi les entrées d'index et informations de circulation dans l'index (cf. §8).



Attention

Les différents types de pages impliqués (cf. Figure 82) ont tous une structure interne qui leur est propre (mais ce sont bien toujours des pages !).

⁶⁵ Ce qui n'est pas forcément vrai dans la pratique...

⁶⁶ Mais aussi de données plus « annexes », toujours pour les relations (cf. §6).

Une BD, gérée par un SGBD, contient donc principalement des relations et, le cas échéant, des index portant sur ces relations. Les relations peuvent être « vues » à 2 niveaux : intensionnel et extensionnel. Le stockage d'une BD demande de stocker principalement :

- Au niveau des relations : des n-uplets,
- Au niveau des index (cf. §8) : des enregistrements d'index (entrées d'index et informations de circulation dans l'index).

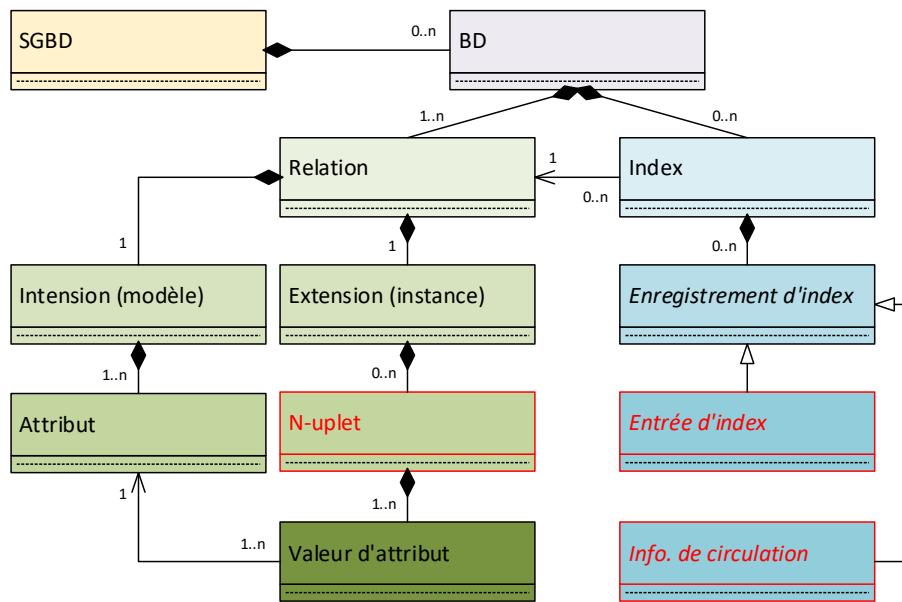


Figure 81. Contenu (grossier) d'une BD

Ces n-uplets (et données annexes, cf. §6) et ces enregistrements d'index (cf. §8) sont stockés dans des pages qui n'ont pas toutes la même organisation interne :

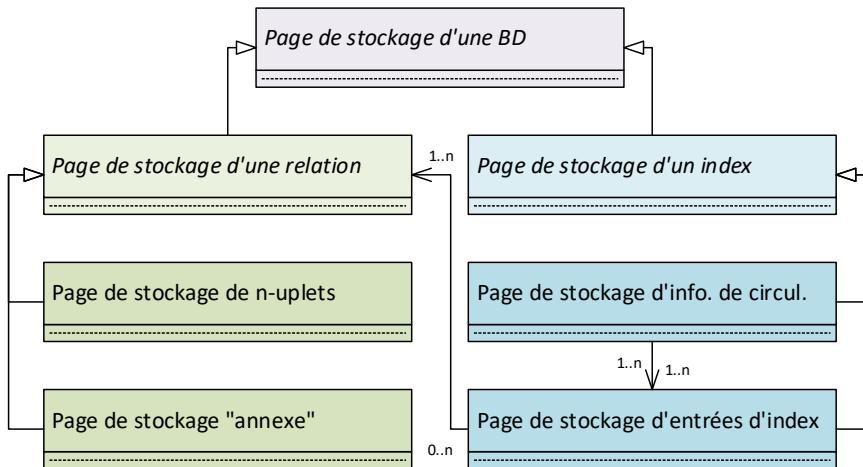


Figure 82. Organisation des pages de stockage des données d'une BD

Le SGBD doit donc permettre :

- De limiter le nombre de pages stockées,
- De limiter le nombre de pages transférées entre mémoire de stockage et mémoire de travail,
- Tout en offrant une facilité d'accès et de gestion suffisante.

5.2 Traitement d'une requête

Usuellement une requête est traitée au travers d'un certain nombre d'étapes. Grossièrement, ce traitement peut se voir comme suit (cf. Figure 83) :

- *L'analyse* : la requête est tout d'abord vérifiée lexicalement, syntaxiquement et sémantiquement (*i.e.* les relations, attributs, ..., indiqués existent-ils comme souhaité ?).
- *Le contrôle* : on vérifie si l'utilisateur demandant à exécuter la requête a les droits d'accès nécessaires pour faire ce qu'il demande de faire sur les données impliquées,
- *L'optimisation* : le moteur d'optimisation établit plusieurs stratégies pour fournir sa réponse à la requête et choisit la stratégie *a priori* la moins coûteuse,
- *L'exécution* : la stratégie retenue est effectivement mise en œuvre par le moteur de requête du SGBD.

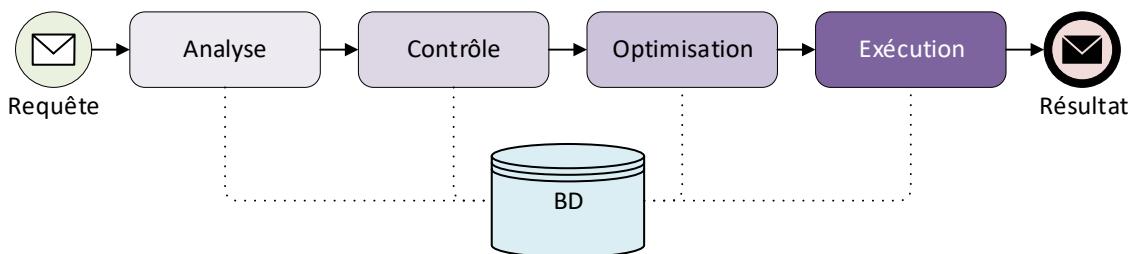


Figure 83. Processus BPMN de traitement d'une requête

Le point le plus critique dans le traitement d'une requête est la performance de l'évaluation de la requête, c'est pourquoi le SGBD met en œuvre des dispositifs dédiés à cette question cruciale des performances d'évaluation.

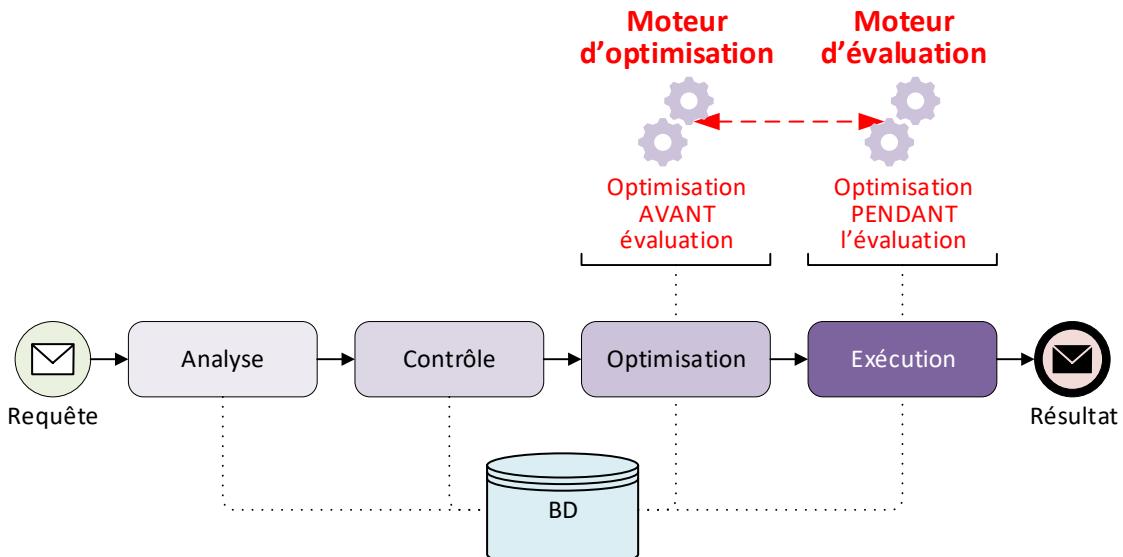


Figure 84. Dispositifs d'optimisation de l'évaluation de requêtes

6 Organisation physique

SOMMAIRE DÉTAILLÉ DU CHAPITRE 6

6.1	Organisation interne générale des pages de stockage des n-uplets	111
6.2	Adressage d'un n-uplet.....	114
6.3	Stockage des données des n-uplets	116
6.4	Stratégies d'adressage des n-uplets	121
6.4.1	Mode d'adressage direct.....	126
6.4.2	Mode d'adressage indirect.....	129
6.4.3	Mode d'adressage direct vs indirect	133
6.5	Stratégies de stockage des valeurs d'attributs	133
6.6	Stratégies de gestion des réertoires des déplacements	135
6.7	Impact des différentes stratégies au niveau des performances	138
6.7.1	Impact du choix du mode d'adressage.....	138
6.7.2	Impact du choix du format de stockage des valeurs d'attributs	139
6.7.3	Impact du choix de gestion du réertoire des déplacements	140
6.8	Paramètres de l'organisation physique	141

FIGURES DU CHAPITRE 6

Figure 88.	Organisation classique d'une page p de stockage de n-uplets.....	111
Figure 89.	Modèle d'organisation d'une page de stockage des n-uplets.....	112
Figure 90.	Déplacement absolu	113
Figure 91.	Déplacements relatifs (en avant ou en arrière)	113
Figure 92.	Données d'un n-uplet	116
Figure 93.	Stockage contigu des données (totales) de n-uplets courts	119
Figure 94.	Stockage « discontinu » de n-uplets au sein d'une page	119
Figure 95.	Stockage « contigu » des données (totales) des n-uplets longs à attributs courts	120
Figure 96.	Stockage « contigu » des données (totales) des n-uplets longs à attribut(s) long(s)	121
Figure 97.	Réduction de la taille d'un n-uplet.....	122
Figure 98.	Agrandissement de la taille d'un n-uplet au sein d'une page pouvant toujours l'accueillir.....	123
Figure 99.	Adressage physique vs adressage logique de n-uplets	125
Figure 100.	Pages de stockage d'une relation en mode d'adressage direct	126
Figure 101.	Mise en œuvre de pointeurs de suivi en mode d'adressage direct	127
Figure 102.	Contenu d'une table de correspondance	129
Figure 103.	Pages de stockage d'une relation en mode d'adressage indirect	131
Figure 104.	Pages d'une table de correspondance vs pages de n-uplets pour une relation R	132

TABLEAUX DU CHAPITRE 6

Tableau 25.	Bilan des stratégies d'adressage des n-uplets.....	133
Tableau 26.	Bilan des stratégies de stockage des valeurs d'attributs.....	134
Tableau 27.	Bilan des stratégies de gestion du réertoire des déplacements.....	136
Tableau 28.	Performances d'accès aux données : paramètres SGBD (génériques)	141
Tableau 29.	Performances d'accès aux données : paramètres SGBD (n-uplets d'une relation R)	142
Tableau 30.	Performances d'accès aux données : paramètres SGBD (PS ou TC d'une relation R).....	143
Tableau 31.	Performances d'accès aux données : nombre « d'informations » par page (d'une relation R).....	144
Tableau 32.	Performances d'accès aux données : nombre de pages occupées (d'une relation R)	145

ÉQUATIONS DU CHAPITRE 6

Équation 4. Taille totale d'un n-uplet	117
Équation 5. Taille des données (brutes) d'un n-uplet.....	117
Équation 6. Taille totale d'un pointeur de suivi.....	128
Équation 7. Taille (fixe) des données (brutes) d'un pointeur de suivi..	128
Équation 8. Nombre de lignes dans une table de correspondance	131
Équation 9. Taille (fixe) d'une ligne d'une table de correspondance.....	131
Équation 10. Taille (totale) d'un n-uplet d'une relation R	138
Équation 11. Adressage direct : impact sur les métadonnées des n-uplets	138
Équation 12. Adressage direct : taille (totale) d'un pointeur de suivi.....	138
Équation 13. Adressage direct : taille des données (brutes) d'un pointeur de suivi.....	138
Équation 14. Adressage direct : taille des métadonnées des pointeurs de suivi	138
Équation 15. Bornage (trivial) du nombre de pointeurs de suivi dans une relation R	138
Équation 16. Taille (fixe) d'une ligne d'une table de correspondance.....	139
Équation 17. Nombre de lignes dans une table de correspondance	139
Équation 18. Format fixe : taille des données (brutes) des n-uplets	139
Équation 19. Format variable : taille des données (brutes) des n-uplets	139
Équation 20. Format variable : impact sur les métadonnées des n-uplets.....	139
Équation 21. Gestion statique : impact sur la place « utile » restant dans une page	140
Équation 22. Gestion dynamique : aucun impact sur la place « utile » dans une page	140
Équation 23. Gestion dynamique : impact sur les métadonnées des n-uplets.....	140
Équation 24. Gestion dynamique (SI adressage direct) : impact sur les métadonnées des pointeurs de suivi.....	140

Nous nous intéressons dans le présent chapitre à l'organisation des pages de stockage des relations. Une relation contenant des n-uplets, nous allons dans un premier temps nous intéresser au stockage de ces derniers (l'essence-même d'une BD !), bien que nous allons avoir besoin de stocker d'autres « informations » (nous allons le voir)...

6.1 Organisation interne générale des pages de stockage des n-uplets

Les pages de stockage des n-uplets⁶⁷ sont classiquement découpées en deux espaces (cf. Figure 85).

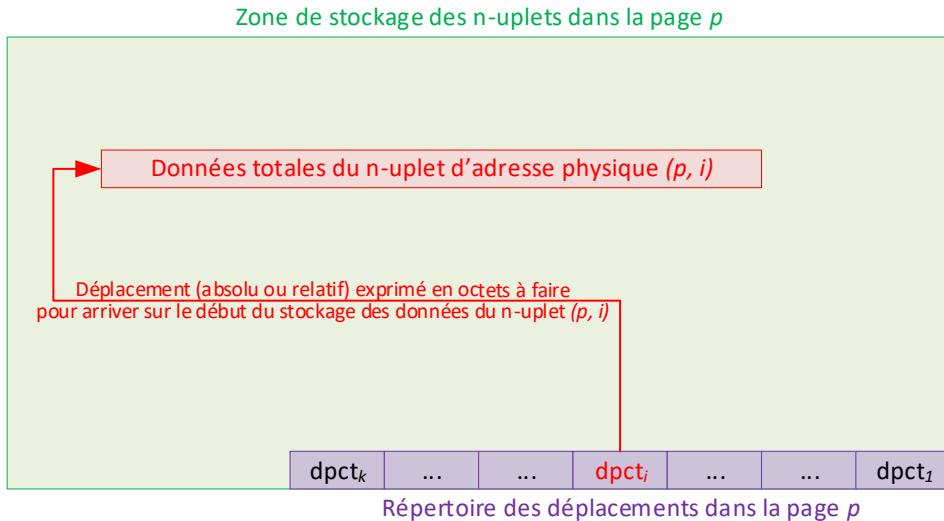


Figure 85. Organisation classique d'une page p de stockage de n-uplets

Ces deux « espaces » sont :

- **La zone de stockage des n-uplets** : les données des n-uplets y sont implantées,
- Un **répertoire des déplacements** qui est implanté à partir de la fin de la page. La $i^{\text{ème}}$ case⁶⁸ de ce répertoire des déplacements indique le déplacement (relatif ou absolu) à faire dans la page pour accéder aux données du n-uplet auquel elle est associée (par convention, on dit qu'il s'agit du n-uplet de **rang i** au sein de la page). **Chaque case du répertoire des déplacements est identifiable de façon unique au sein du répertoire des déplacements de « sa » page par son identificateur (i.e. son indice) $id_{caseDépl}$.**



Question

Que-ce que c'est que cette histoire de « déplacement » ?

Réponse

Savoir dans quelle page se trouve un n-uplet n'est pas suffisant : encore faut-il savoir où dans cette page le stockage de ce n-uplet débute (à quel octet) !

⁶⁷ Le concept de « page » étant sous-jacent à toute donnée dans un SGBD, on peut bien sûr stocker dans une page bien autre chose que des n-uplets (des index par exemple). Elles ne sont alors pas forcément organisées de la même façon.

⁶⁸ Par convention, la première position est la position 1.



Attention

L'identificateur $id_{caseDépl}$ d'une case d'un répertoire des déplacements n'a AUCUNE raison d'être stocké dans ladite case (de la même façon que, en programmation, l'indice d'une case d'un tableau n'est PAS stocké dans cette case) : ces identificateurs de cases $id_{caseDépl}$ sont en réalité gérés « ailleurs » (par le SGBD).



Remarque

Avec une telle organisation, un n-uplet contenu dans une page p est **forcément** associé à une case du répertoire des déplacements implanté à la fin de cette page p , case contenant le déplacement à réaliser pour arriver aux données de ce n-uplet.

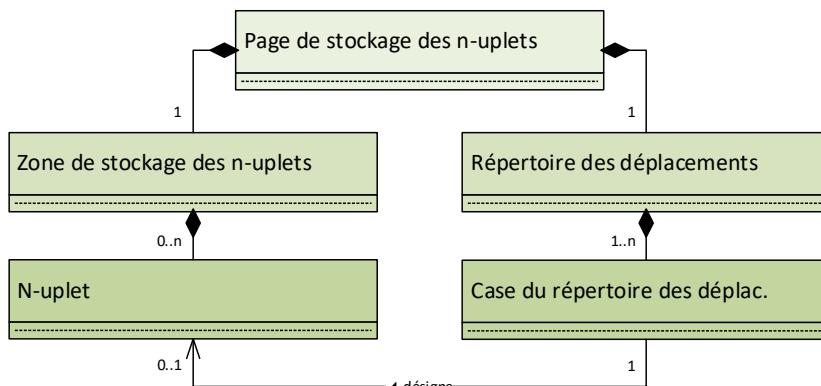


Figure 86. Modèle d'organisation d'une page de stockage des n-uplets

Ainsi, si on dispose en mémoire de travail de la page id_{page} contenant le n-uplet que l'on souhaite manipuler ET si on connaît l'indice $id_{caseDépl}$ de la case du répertoire des déplacements auquel ce n-uplet est « associé », manipuler ce n-uplet revient simplement à :

1. Lire dans le répertoire des déplacements implanté à la fin de cette page la valeur du déplacement contenu dans la case d'indice $id_{caseDépl}$,
2. Effectuer ce déplacement au sein de la page (voir ci-dessous) : on est alors au début de la zone précise de stockage du n-uplet cherché au sein de la page qui le contient.



Définition : « déplacement »

Un **déplacement** est une valeur (entier, strictement positif ou non selon les usages, indiquant un nombre d'octets) permettant un « mouvement » à réaliser dans une zone mémoire depuis un octet donné (point de départ) de cette zone mémoire vers un autre octet de cette même zone mémoire (point d'arrivée). Le déplacement peut alors être absolu (le déplacement a alors une valeur forcément positive ou nulle) ou relatif (le déplacement a alors une valeur négative s'il doit se faire « en arrière », nulle ou positive s'il doit se faire « en avant »⁶⁹).

⁶⁹ Encore faut-il définir ce que sont l'avant et l'arrière selon le cas...

Opérer un déplacement, comme indiqué ci-dessus, peut se faire de 2 manières :

- *Déplacement absolu* (cf. Figure 87) : la donnée à laquelle on souhaite accéder se trouve à l'octet indiqué par la valeur du déplacement en partant du début de la « zone mémoire » considérée,

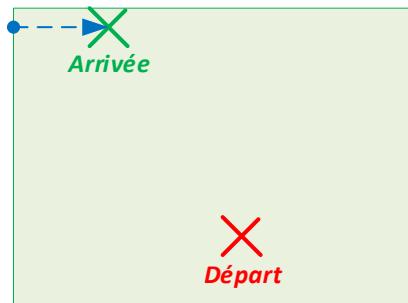


Figure 87. Déplacement absolu

- *Déplacement relatif* (cf. Figure 88) : la donnée à laquelle on souhaite accéder est atteinte en avançant ou en reculant (selon l'usage du déplacement) d'un nombre d'octets égal à la valeur du déplacement depuis « là où on est » dans la « zone mémoire » considérée.

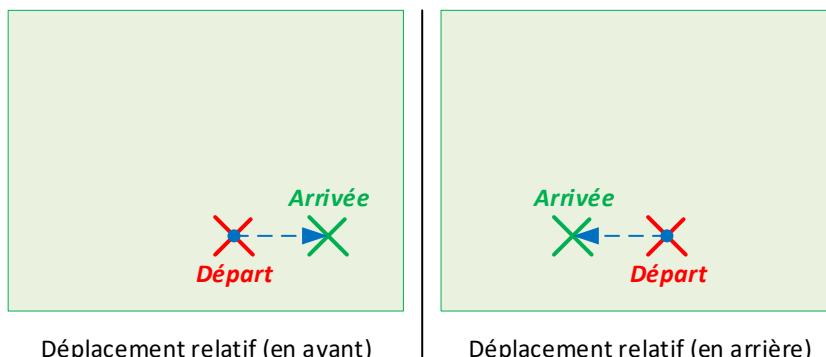


Figure 88. Déplacements relatifs (en avant ou en arrière)



Remarque

Dans le cadre de la gestion des pages de stockage des n-uplets, les déplacements, s'ils sont relatifs, ne se font qu'en arrière (puisque'ils partent alors d'une case du répertoire des déplacements pour « revenir » vers le début du n-uplet associé).



Question

Mieux vaut-il faire les déplacements de façon absolue ou de façon relative ?

Réponse

Étant donné que nous avons considéré que les temps de traitement au sein de la mémoire de travail étaient négligeables, le choix importe finalement peu. Cependant, dans l'optique de simplifier le SGBD, **si on choisit l'une des 2 stratégies de réalisation des déplacements, c'est la même qui sera utilisée pour toute la BD⁷⁰**.

⁷⁰ Voir même pour toutes les BD gérées par le même SGBD.

Ainsi, l'indice de la case du répertoire des déplacements associée à chaque n-uplet permet, combinée avec d'autres identifiants, d'identifier de façon unique un n-uplet de n'importe quelle BD gérée par un SGBD. Cet adressage permet également de savoir où est stocké physiquement le n-uplet sur la mémoire de stockage.

6.2 Adressage d'un n-uplet

Il est donc possible d'identifier de façon unique un n-uplet sur une mémoire de stockage : cet identifiant unique s'appelle une **adresse** (et on parle « d'adressage »).



Définition : « adresse (d'une donnée) »

Une **adresse** permet non seulement d'identifier de façon unique une donnée parmi toute autre donnée mais aussi de pouvoir accéder à cette donnée (pour la lire, la manipuler ou l'écrire).

Dans le cas des SGBD, l'adresse d'un n-uplet identifie donc chaque n-uplet de façon unique ET permet aussi d'accéder physiquement à la zone de stockage où il est enregistré en mémoire de stockage.



Question

Et pour les accès en mémoire de travail ?

Réponse

C'est l'OS qui gère cela : il possède un mécanisme permettant de connaître à tout instant quelle page de la mémoire de stockage est présente en mémoire de travail et, le cas échéant, où elle se trouve en mémoire de travail. Donc, si on sait qu'un n-uplet se trouve dans une page id_{page} en mémoire de stockage, l'OS sait si cette page est déjà présente (*i.e.* si elle a été chargée) en mémoire de travail et, si oui, où elle se trouve au sein de la mémoire de travail. **Un seul adressage suffit donc : celui des données au sein de la mémoire de stockage (l'adressage en mémoire de travail étant donc géré de façon transparente par l'OS).**

Si on reprend tout ce que l'on a dit précédemment (cf. §5.1 et §6.1), accéder à un n-uplet particulier nécessite de connaître et d'indiquer :

- L'identificateur id_{site} du site sur lequel il est stocké,
- L'identificateur $id_{fichier}$ du fichier (au sein du site id_{site}) qui le contient,
- L'identificateur id_{bloc} du bloc (contenant tout ou partie du fichier $id_{fichier}$) qui le contient,
- L'identificateur id_{page} de la page (au sein du bloc id_{bloc}) qui le contient,
- L'identificateur $id_{caseDépl}$ de la case du répertoire des déplacements (au sein de la page id_{page}) qui contient le déplacement à réaliser pour atteindre le n-uplet dans la page id_{page} .



Définition : « adresse (théorique) d'un n-uplet »

L'**adresse d'un n-uplet** est un quintuplet $(id_{site}, id_{fichier}, id_{bloc}, id_{page}, id_{caseDépl})$. Cette adresse (forcément unique pour chaque n-uplet) permet d'identifier un n-uplet parmi tous et de savoir où il est enregistré (et, donc, permet d'y accéder).

Avec un tel adressage, accéder à un n-uplet donné se fait grossièrement comme suit :

1. S'il n'est pas déjà présent en mémoire de travail, charger depuis la mémoire de stockage le bloc id_{bloc} (contenant tout ou partie du fichier $id_{fichier}$) situé sur le site id_{site} ,
2. Accéder à la page id_{page} du bloc id_{bloc} maintenant présent en mémoire de travail,
3. Lire dans le répertoire des déplacements implanté à la fin de cette page la valeur du déplacement contenu dans la case d'indice $id_{caseDépl}$,
4. Effectuer ce déplacement au sein de la page : on est alors au début de la zone précise de stockage du n-uplet cherché au sein de la page qui le contient.



En pratique

On peut « interpréter » une adresse de n-uplet (id_{site} , $id_{fichier}$, id_{bloc} , id_{page} , $id_{caseDépl}$) comme suit : elle désigne le n-uplet de « rang » $id_{caseDépl}$ au sein de la page id_{page} contenue dans le bloc id_{bloc} du fichier $id_{fichier}$ stocké sur le site id_{site} .

Pour simplifier, nous allons adopter 3 hypothèses :

- On ne traitera que des BD mono-sites \Rightarrow on n'a donc pas besoin d'un identificateur de site id_{site} ,
- On ne traitera que des BD mono-fichiers \Rightarrow l'identificateur de fichier $id_{fichier}$ est donc inutile,
- On suppose que la taille d'une page est exactement celle d'un bloc (donc qu'un bloc ne contient qu'une seule page) \Rightarrow l'identificateur d'un bloc id_{bloc} est équivalent à l'identificateur id_{page} de l'unique page qu'il contient.

En tenant compte de ces 3 hypothèses simplificatrices, l'adresse identifiant un n-uplet peut alors se réduire à un couple (id_{page} , $id_{caseDépl}$)⁷¹. Nous ferons la même supposition dans toute la suite du cours (i.e. dans le présent chapitre mais aussi dans les suivants). Ainsi, les n-uplets seront toujours identifiés par un couple (id_{page} , $id_{caseDépl}$) où id_{page} est l'identifiant de la page dans laquelle ils se trouvent et $id_{caseDépl}$ est leur rang dans cette page (i.e. l'indice de la case qui leur est dédiée dans le répertoire des déplacements implanté à la fin de la page id_{page}).



Définition : « adresse (simplifiée) d'un n-uplet »

En tenant compte de nos 3 hypothèses simplificatrices (voir ci-dessus), l'**adresse (simplifiée) d'un n-uplet** est un couple (id_{page} , $id_{caseDépl}$). Cette adresse (forcément unique pour chaque n-uplet) permet d'identifier un n-uplet parmi tous et de savoir où il est enregistré (et, donc, permet d'y accéder).

⁷¹ Ou à un couple (id_{bloc} , $id_{caseDépl}$), puisque selon notre 3^{ème} hypothèse $id_{bloc} \Leftrightarrow id_{page}$, mais nous avons choisi la première possibilité.



Attention

Ces hypothèses simplificatrices impactent aussi certains des paramètres liés à la gestion de la mémoire (stockage et travail), notamment :

- La taille d'un bloc et d'une page sont les mêmes (notez que, du coup, la taille d'un bloc devient aussi implicitement indépendante d'une mémoire) :

$$T_{page}^{fixe}(OS) = T_{bloc}^{fixe}(Mem, OS)$$

- **On fera des chargements et des déchargements de pages depuis/vers la mémoire de stockage** (leurs temps d'exécution sera égale à celui des blocs, toujours considéré comme négligeable, i.e. nul, si la mémoire considérée est une mémoire de travail) :

$$tps_{lectPage}^{fixe}(Mem, OS) = tps_{lectBloc}^{fixe}(Mem, OS)$$

$$tps_{ecritPage}^{fixe}(Mem, OS) = tps_{ecritBloc}^{fixe}(Mem, OS)$$

Avec cet adressage simplifié, accéder à un n-uplet donné se fait grossièrement comme suit :

1. Si elle n'est pas déjà présente en mémoire de travail, charger depuis la mémoire de stockage la page id_{page} ,
2. Lire dans le répertoire des déplacements implanté à la fin de cette page la valeur du déplacement contenu dans la case d'indice $id_{caseDépl}$,
3. Effectuer ce déplacement au sein de la page : on est alors au début de la zone précise de stockage du n-uplet cherché au sein de la page qui le contient.

Reste maintenant à interpréter ce que contient cette zone de stockage des données d'un n-uplet au sein de « sa » page...

6.3 Stockage des données des n-uplets

Lorsqu'on parle des données d'un n-uplet, on pense immédiatement aux valeurs que possède ce n-uplet pour chaque attribut défini dans le modèle de la relation à laquelle il appartient. Cela n'est que partiellement exact... Nous allons le voir dans cette partie mais, en réalité, lorsque l'on parle de « données contenues » dans un n-uplet, on ne parle pas uniquement des valeurs des attributs de ce n-uplet : le SGBD stocke aussi d'autres informations (métadonnées). On parle alors :

- *De données brutes* : là, on parle bien des valeurs des attributs du n-uplet considéré,
- *De métadonnées* : il s'agit d'informations que le SGBD associe à chaque n-uplet,
- *De données totales* : il s'agit des données brutes et de métadonnées que le SGBD leur associe.

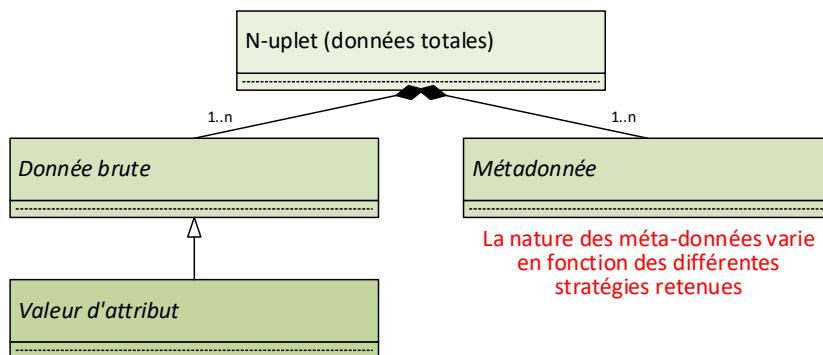


Figure 89. Données d'un n-uplet

Définition : « données d'un n-uplet »

La notion de **données d'un n-uplet** est usuellement souvent imprécise. Elle peut en fait correspondre...

- *Aux données brutes* : les valeurs des attributs du n-uplet considéré,
- *Aux métadonnées* : les informations associées par le SGBD à ce n-uplet⁷²,
- *Aux données totales* : l'ensemble des données brutes et des métadonnées.

Ainsi :



$$T_{totaleNU}^{fixe|moy|min|max}(R) = T_{donneesNU}^{fixe|moy|min|max}(R) + T_{metaNU}^{fixe}(R)$$

Équation 4. Taille totale d'un n-uplet

Avec (si le n-uplet appartient à la relation R) :

$$T_{donneesNU}^{fixe|moy|min|max}(R) = \sum_{i=1}^{i \leq nb_{attributs}^{fixe}(R)} T_{valAttr}^{fixe|moy|min|max}(R, att_i)$$

Équation 5. Taille des données (brutes) d'un n-uplet

La taille des métadonnées d'un n-uplet dépend des différentes stratégies choisies pour le stockage des n-uplets (mais elle est fixe une fois ces stratégies fixées).

Remarque

Usuellement, les métadonnées comprennent (voir plus bas) :

- Un éventuel drapeau (0 ou 1 pour chaque n-uplet, selon la stratégie de stockage choisie pour les données des n-uplets, cf. §6.4),
- Éventuellement des tailles pour les valeurs d'attributs (0 ou n pour chaque n-uplet, n étant le nombre d'attributs de la relation à laquelle appartient le n-uplet, selon la stratégie de stockage choisie pour les valeurs d'attributs, cf. §6.5),
- Éventuellement la case du répertoire des déplacements associée à ce n-uplet (selon la stratégie de gestion choisie pour le répertoire des déplacements, cf. §6.6).

Afin de simplifier le stockage des données des n-uplets, un principe simple est adopté : les données d'un même n-uplet (données brutes et métadonnées) sont, sauf cas particuliers, stockées de façon contigüe (*i.e.* les unes à la suite des autres) dans la page contenant le n-uplet.



Question

Quels sont ces cas particuliers ?

Réponse

Forcément, si un n-uplet ne tient pas complètement dans une page⁷³, il est « difficile » de ranger ses données dans cette page de façon contigüe !!!

Cette réponse amène une seconde question...

⁷² Elles dépendent de différentes stratégies choisies (dont la taille de ces métadonnées aussi).

⁷³ En réalité, dans la place occupée par la zone de stockage des n-uplets au sein d'une page.

Question

Pourquoi un n-uplet ne tiendrait pas complètement dans une page⁷³ ? On peut raisonnablement imaginer qu'elles sont assez grandes pour en contenir au moins un, non ?



Réponse

C'est raisonnable mais pas forcément vrai ! En effet, un n-uplet peut être plus grand qu'une page⁷³ pour 2 raisons (cumulables l'une avec l'autre) :

- Parce que la valeur de l'un (ou plusieurs) de ses attributs est elle-même « très grosse » : les SGBD modernes permettent par exemple de stocker des fichiers (sous forme de binaires) en tant que valeur d'attributs ; ces binaires ont très souvent une taille importante (typiquement dans le cas de vidéos) qui dépasse la taille d'une page⁷³,
- Parce que le modèle (*i.e.* l'intension) de la relation à laquelle ce n-uplet appartient définit un très grand nombre d'attributs : même si, pris indépendamment les uns des autres, leurs valeurs respectives occupent une petite place⁷⁴, le nombre des valeurs d'attributs à stocker est du coup très grand ; mises bout à bout, ces valeurs d'attributs occupent donc une grande place, parfois supérieure à celle d'une page⁷³.

En plus de cela, il ne faut pas oublier de prendre en compte la place occupée par les métadonnées des n-uplets !

On peut alors classer les n-uplets en plusieurs catégories selon leur taille totale par rapport à celle de la page⁷⁵ dans laquelle ils se trouvent. Ainsi, on considère :

- *Les n-uplets courts (cas le plus courant)* : ce sont les n-uplets dont les données (totales) occupent une place inférieure à la taille de la zone de stockage des n-uplets de la page dans laquelle ils se trouvent⁷⁶ ; les données (totales) des n-uplets courts sont toujours stockées de façon contigüe dans la zone de stockage de la page dans laquelle ils se trouvent (cf. Figure 90). Par souci de simplicité, **les données totales d'un n-uplet court ne sont JAMAIS stockées à cheval sur plusieurs pages : elles sont toujours intégralement contenues dans une seule page.**

⁷⁴ Et encore, certaines peuvent occuper une « grande place » (mais toujours inférieure à la taille de la page).

⁷⁵ Pour simplifier et homogénéiser (puisque la taille de la zone de stockage des n-uplets des différentes pages d'un SGBD n'est en réalité PAS forcément identique d'une page de stockage de n-uplets à l'autre, principalement en raison de la place, parfois variable, occupée par le répertoire des déplacements), on compare souvent à la taille d'une page (cette dernière étant fixe pour toutes les pages du SGBD, aussi bien pour celles qui servent au stockage des n-uplets que pour les autres dont nous parlerons plus loin).

⁷⁶ Pour simplifier, on parle souvent de n-uplets courts quand la taille de leurs données est inférieure à la taille d'une page, la place occupée par le répertoire des déplacements à la fin de cette page étant alors considérée comme négligeable.

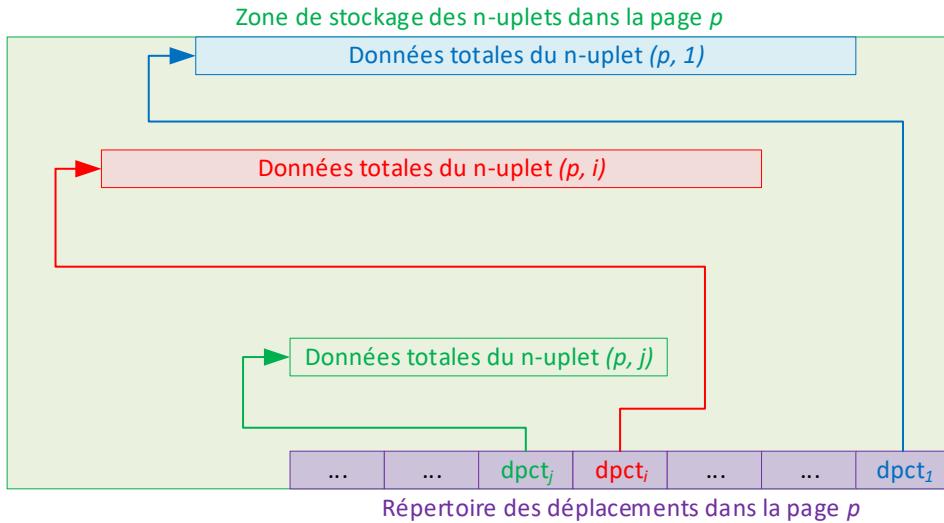


Figure 90. Stockage contigu des données (totales) de n-uplets courts

Remarque

En revanche, on n'oblige pas le stockage contigu de données (totales) de plusieurs n-uplets : qu'il y ait des « trous » dans la zone de stockage importe peu, le temps de réorganisation des données contenues dans une même page étant considéré comme négligeable puisque réalisé en mémoire de travail.

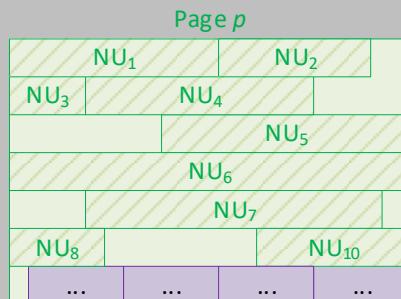


Figure 91. Stockage « discontinu » de n-uplets au sein d'une page

- *Les n-uplets longs dont les données ne tiennent pas dans une seule page* : on peut distinguer...
 - *Les n-uplets longs dont tous les attributs sont courts* : ce sont les n-uplets dont la valeur de chacun des attributs, pris indépendamment les uns des autres, tient dans la zone de stockage d'une page⁷⁷ ; comme pour les n-uplets courts, les données sont stockées de façon contigüe mais, dès qu'une valeur d'attribut ne rentre pas sur la page courante, on insère un code « spécial » (code de continuité de n-uplet), contenant entre autres choses l'identificateur p' de « la page suivante » associé à un rang i' , et les valeurs des attributs suivants sont stockés sur la page p' (puis éventuellement p'' , p''' , ..., si nécessaire, cf. Figure 92) en tenant compte du déplacement indiqué dans la case i' du répertoire des déplacements de cette page p' . Toujours par souci de simplicité de gestion, **la valeur d'un attribut court n'est JAMAIS stockée à cheval sur plusieurs pages : elle est toujours intégralement contenue dans une seule page**. Enfin, les attributs courts d'un même n-uplet présents sur une page donnée sont stockés de façon contigüe au sein de cette page.

⁷⁷ Toujours par simplicité, on dira « dans une page ».



Remarque

Bien sûr, le code « spécial » de poursuite du stockage des attributs d'un n-uplet (code de continuité de n-uplet) est choisi par l'éditeur du SGBD de façon qu'il ne soit pas rencontré par ailleurs⁷⁸.

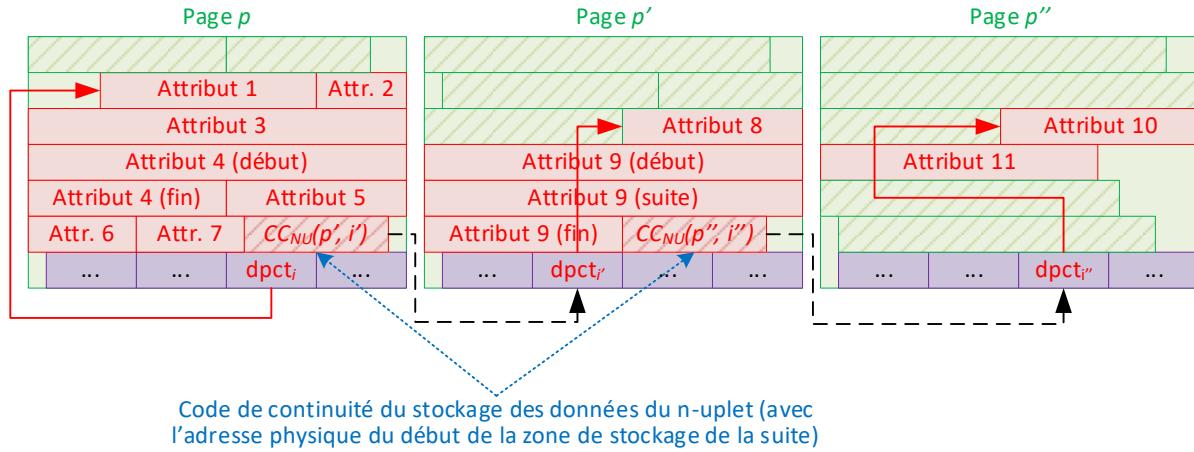


Figure 92. Stockage « contigu » des données (totales) des n-uplets longs à attributs courts

- *Les n-uplets longs dont au moins un des attributs est long* : ce sont les n-uplets dont la valeur d'au moins un des attributs, pris indépendamment les uns des autres, ne tient pas dans la zone de stockage d'une page⁷⁹ ; la liste des attributs est souvent réorganisée⁸⁰ afin de débuter le stockage des attributs du n-uplet par les attributs courts puis de stocker les attributs longs ; là encore, un autre code « spécial » (code de continuité d'attribut), contenant entre autres choses l'identificateur p' de « la page suivante » associé à un rang i' , est utilisé pour indiquer que la valeur d'un attribut long continue sur la page p' et que la suite du stockage débute en effectuant le déplacement indiqué dans la case i' (cf. Figure 93).



Remarque

Bien sûr, là encore, le code « spécial » de poursuite du stockage de la valeur d'un attribut long (code de continuité d'attribut) est choisi par l'éditeur du SGBD de façon qu'il ne soit pas rencontré par ailleurs⁸¹. Ce code est bien sûr différent de celui qui indique la poursuite du stockage d'un n-uplet (code de continuité de n-uplet).

⁷⁸ La probabilité de rencontrer ce code étant statistiquement trop faible pour être considérée comme une éventualité.

⁷⁹ Toujours par simplicité, on dira « dans une page ».

⁸⁰ Cette réorganisation est alors « consignée » dans les « tables systèmes » que le SGBD établit pour chaque BD. Elle n'est cependant réalisée que si c'est nécessaire (mais c'est souvent le cas !).

⁸¹ La probabilité de rencontrer ce code étant statistiquement trop faible pour être considérée comme une éventualité.

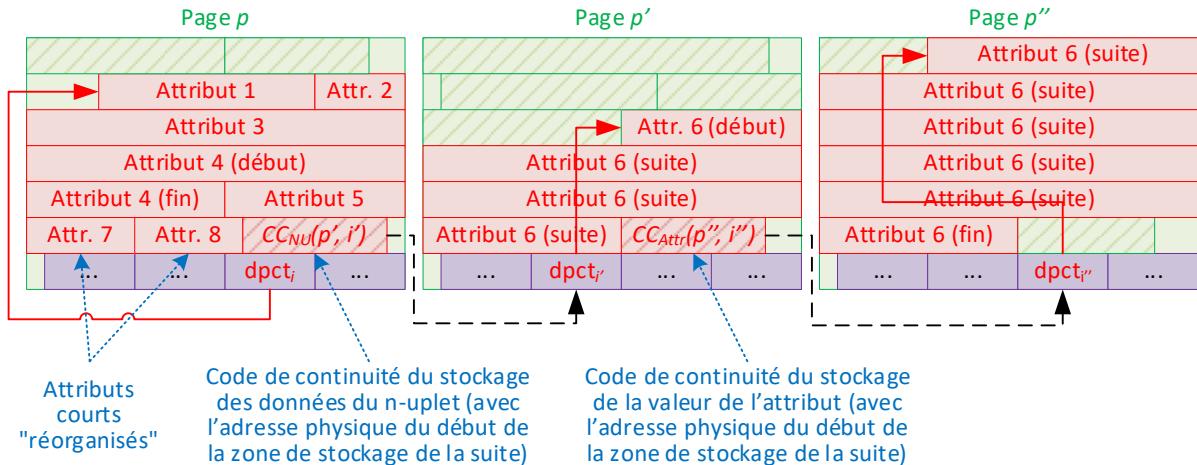


Figure 93. Stockage « contigu » des données (totales) des n-uplets longs à attribut(s) long(s)

6.4 Stratégies d'adressage des n-uplets

Nous en avons déjà parlé, le stockage physique des données doit combiner plusieurs avantages, pourtant parfois antagonistes : la place occupée doit être raisonnable, les performances d'accès (lecture et écriture) doivent être convenables et, enfin, la gestion du stockage physique doit être la plus simple possible (précisément pour ne pas grever les performances d'accès).

Par rapport au dernier point (*i.e.* la simplicité de la gestion du stockage), nous avons déjà indiqué quelques règles d'uniformisation (les tailles des secteurs, blocs et pages sont fixes, les n-uplets courts ne sont jamais stockés à cheval sur 2 pages, les valeurs d'attributs courts ne sont jamais stockées à cheval sur 2 pages, ...). Cependant, le point le plus important pour simplifier la gestion de stockage est le suivant : **l'adresse associée à un n-uplet ne doit PAS varier durant toute son existence**. L'idée qui est derrière est la suivante : **une fois qu'un n-uplet est « connu » (*i.e.* identifié) dans le SGBD par une adresse, cette connaissance doit rester pérenne⁸²**.



Question

Mais pourquoi voudrait-on/devrait-on modifier l'adresse (id_{page} , $id_{caseDépl}$) identifiant un n-uplet ?

Réponse

Les n-uplets « vivent » dès lors que l'on manipule la BD qui les contient, ce qui peut notamment entraîner une modification de leur taille. Et, puisque cette taille peut varier, on ne peut PAS garantir de toujours pouvoir stocker un n-uplet dans sa page d'origine id_{page} (associé à la case d'indice $id_{caseDépl}$ du répertoire des déplacements situé à la fin de cette page).

⁸² Cette pérennité de l'adresse par laquelle un n-uplet est connu tout au long de sa vie participe à l'indépendance données/traitements.

Les principales opérations pouvant entraîner une modification de la taille d'un n-uplet sont :

- *La modification de la valeur d'un ou de plusieurs attributs d'un n-uplet (cas usuel)* : si on est en format fixe de stockage des valeurs d'attributs (cf. §6.5), pas de souci, mais si on est en format variable (cf. §6.5), alors la valeur des attributs modifiés (et, donc, le n-uplet) peut être réduite ou, au contraire, être agrandie,
- *La modification du schéma conceptuel de la relation à laquelle le n-uplet appartient (cas rare)* : une telle modification se fait souvent par ajout ou suppression d'attributs à la relation ou alors par modification du type d'un ou de plusieurs attributs de la relation ; dans de tels cas, que l'on soit en format fixe ou variable pour le stockage des valeurs d'attributs (cf. §6.5), la taille des n-uplets de la relation peut être modifiée (réduite ou agrandie).

Ainsi :

- *Si la taille du n-uplet après modification reste inchangée* : pas de souci, le n-uplet reste « où il est » dans la page p et ni son adresse (p, i) ni le déplacement contenu dans la case d'indice i du répertoire des déplacements de la page p n'ont besoin d'être modifiés au niveau de l'adresse du n-uplet,
- *Si la taille du n-uplet est réduite* : là encore, pas de souci, le n-uplet peut rester dans la même page p associé à la même case d'indice i dans le répertoire des déplacements de cette page étant donné que l'on sait que la zone de stockage de cette page contient suffisamment de place pour contenir le n-uplet après sa modification⁸³ (cf. Figure 94),

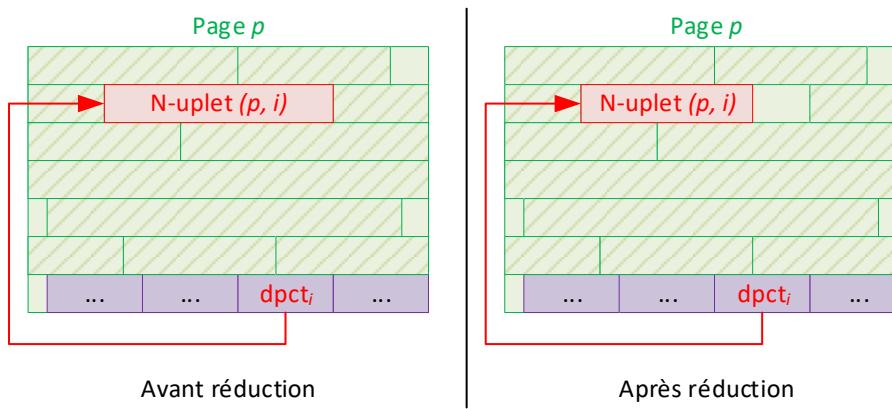


Figure 94. Réduction de la taille d'un n-uplet

- *Si la taille du n-uplet est agrandie* : ce cas est forcément moins trivial que le précédent ; on peut en fait le « décomposer » en 2 sous-cas...
 - *S'il reste suffisamment de place dans la zone de stockage de cette page p* (quitte à la « défragmenter » au préalable) pour accueillir le n-uplet après son « agrandissement », le n-uplet reste dans la page p et est toujours associé à la même case d'indice i du répertoire de cette page (seul le déplacement contenu dans cette case est modifié, cf. Figure 95).

⁸³ Il est même possible, afin d'optimiser le niveau de fragmentation de la zone de stockage de cette page p , de déplacer le n-uplet en son sein sans avoir à changer de page p ni avoir à changer l'indice i de la case du répertoire des déplacements à laquelle est associé le n-uplet : son adresse (p, i) reste donc la même, seule la valeur du déplacement contenue dans la case d'indice i du répertoire des déplacements est modifiée.

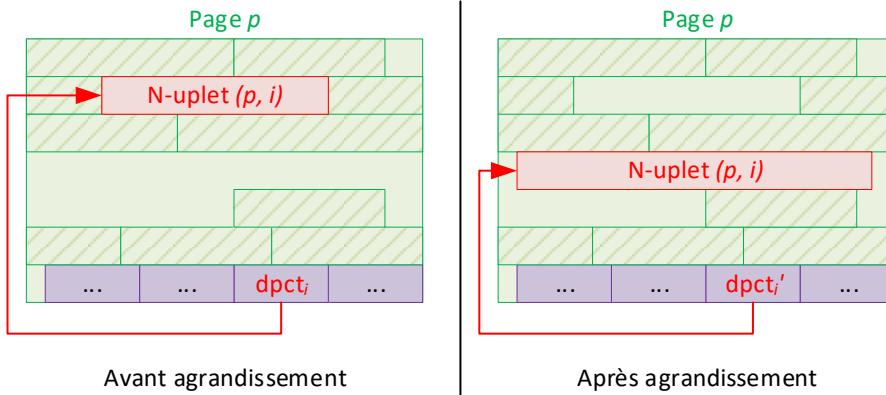


Figure 95. Agrandissement de la taille d'un n-uplet au sein d'une page pouvant toujours l'accueillir

- *S'il ne reste pas suffisamment de place libre dans la zone de stockage de la page p (même si on faisait préalablement une « défragmentation » de cette zone) pour accueillir le n-uplet après son « agrandissement », on n'a pas le choix : il va falloir transférer le n-uplet de sa page actuelle p vers une nouvelle page p' et, implicitement, son adresse actuelle (p, i) deviendrait (p', i') , ce qui ne respecte pas la contrainte énoncée plus haut (*i.e.* « l'adresse identifiant un n-uplet ne doit pas varier durant toute son existence »).*

Question

Pourquoi ne pas laisser le n-uplet là où il est (*i.e.* dans sa page d'origine p), sans changer son adresse (p, i) , et utiliser un code de continuité de n-uplet ou d'attribut pour poursuivre son stockage sur une autre page ?



Réponse

Tout dépend de la nature du n-uplet après son agrandissement :

- S'il est devenu (ou resté) un n-uplet long, pas de souci, on peut effectivement faire comme cela (et c'est d'ailleurs fait comme cela) : on retombe de toutes les façons sur ce qui est fait pour stocker un n-uplet long (qu'il ait été long dès sa création ou après une modification importante peu),
- **S'il est resté un n-uplet court, on ne peut PAS faire comme cela : rappelons que, par principe, un n-uplet court n'est JAMAIS stocké à cheval sur 2 pages !**

Question

Mais, dans ce cas, on est bien obligé de changer le n-uplet de page, donc son adresse. Comment respecter la contrainte qui veut que l'adresse d'un n-uplet ne change pas tout au long de son existence ? C'est impossible du coup !



Réponse

En mettant en œuvre un mécanisme d'adresses intermédiaires.

Le principe adopté est le suivant : en fait, un n-uplet est associé à 2 adresses, toutes 2 l'identifiant de façon unique dans le SGBD ! L'une de ces adresses restera invariable durant toute la vie du n-uplet mais l'autre pourra, elle, être modifiée. On distingue alors pour chaque n-uplet :

- *Son adresse physique (réelle)* : elle indique la zone de stockage du n-uplet en mémoire de stockage⁸⁴ ; elle pourra varier dans le temps (en cas de déplacement d'un n-uplet d'une page p vers une page p' , comme nous venons de le voir).
- *Son adresse logique (virtuelle)* : elle n'a pas forcément de lien avec la zone de stockage du n-uplet, elle sert « juste » à l'identifier de façon unique au sein du SGBD ; cette adresse logique restera, elle, invariante dans le temps.

L'idée est alors la suivante :

- L'adresse physique (réelle), toujours de la forme (p, i) , est attribuée au n-uplet à sa création et pourra varier pendant sa durée de vie,
- L'adresse logique (virtuelle), dont la forme dépend de la stratégie d'adressage choisie, est également attribuée au n-uplet à sa création et ne variera pas durant tout sa durée de vie,
- Un « mécanisme » est mis en place au niveau du gestionnaire des accès aux données afin de permettre à tout instant de lier l'adresse logique à l'adresse physique d'un même n-uplet : les « composants » du SGBD connaissent donc le n-uplet par son adresse logique, invariante mais virtuelle, et ce « mécanisme » fait le lien avec son adresse physique, variante mais réelle, afin d'accéder effectivement aux données de ce n-uplet.



Définition : « adresse physique (réelle) d'un n-uplet »

L'**adresse physique (réelle) d'un n-uplet** est de la forme (p, i) . Elle identifie donc de façon unique ce n-uplet et indique effectivement la zone sur la mémoire de stockage où sont enregistrées les données de ce n-uplet (p est l'identifiant de la page concernée et i le « rang » du n-uplet dans cette page, *i.e.* l'indice de la case du répertoire des déplacements de cette page contenant le déplacement à réaliser pour trouver le 1^{er} octet de stockage des données du n-uplet dans la page p). L'adresse physique d'un n-uplet peut, si besoin, être changée durant la vie de ce n-uplet.



Définition : « adresse logique (virtuelle) d'un n-uplet »

L'adresse logique (virtuelle) d'un n-uplet identifie, elle aussi, de façon unique le n-uplet dans le SGBD. **Cependant, contrairement à l'adresse réelle, elle n'indique pas forcément la zone sur la mémoire de stockage où sont enregistrées les données de ce n-uplet.** Pour accéder à ces données, un mécanisme (supporté par le gestionnaire des accès aux données), dépendant de la stratégie d'adressage choisie, permet de relier l'adresse logique à l'adresse réelle et, donc, à la zone où sont effectivement stockées les données du n-uplet. L'adresse logique d'un n-uplet est invariante durant sa vie.

Ainsi, on décorrèle la notion « d'identifiant unique d'un n-uplet » et celle « d'indicateur de la zone de stockage des données du n-uplet » :

- L'adresse physique (réelle) indique la zone de stockage des données du n-uplet et peut varier dans le temps si besoin,
- L'adresse logique (virtuelle) identifie de façon unique le n-uplet au sein du SGBD et est invariant pendant toute la vie du n-uplet,
- La stratégie d'adressage choisie, met en place au niveau du gestionnaire d'accès aux données un « mécanisme » pour relier ces 2 notions.

⁸⁴ Donc, aussi en mémoire de travail (*via* les mécanismes dédiés de l'OS).

Le n-uplet est donc « connu » par les différents moteurs/services/gestionnaires du SGBD (sauf celui d'accès aux données, donc) par son adresse logique (virtuelle), invariante dans le temps. C'est le gestionnaire d'accès aux données qui se charge de « passer » de cette adresse logique à l'adresse physique (réelle) actuelle du n-uplet afin de le lire ou l'écrire. Ce « passage » de l'adresse logique vers l'adresse physique se fait en tenant bien sûr compte de la stratégie d'adressage retenue.

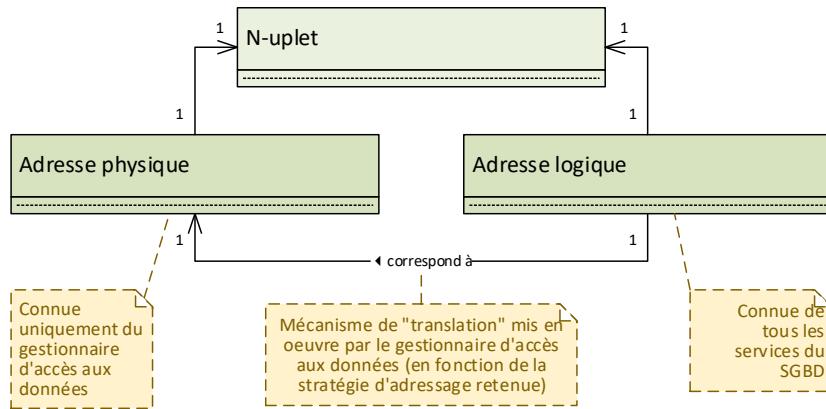


Figure 96. Adressage physique vs adressage logique de n-uplets



Remarque

Parfois, selon la stratégie choisie, ces 2 adresses d'un même n-uplet (physique et logique) peuvent être égales (mais pour certains n-uplets à certains moments de leur vie, notamment à leur création).

Pour faire le lien entre ces 2 adresses (réelle et virtuelle) d'un n-uplet, on paramètre le gestionnaire d'accès aux données en choisissant une stratégie d'adressage : on parle de « **mode d'adressage** ». Il en existe 2 principales⁸⁵ (avec, bien sûr, des stratégies dérivées plus ou moins complexes et performantes) :

- Le mode d'adressage direct,
- Le mode d'adressage indirect.



Remarque

Pour être plus parlant, nous allons principalement illustrer les 2 stratégies d'adressage présentées ci-après au travers du cas où le n-uplet, par suite d'un agrandissement, doit être changé de page (donc le cas où son adresse physique initiale (p, i) devient (p', i')).

⁸⁵ Il existe aussi un mode d'adressage mixte (ce dernier peut être vu comme « une combinaison » des stratégies d'adressage directe et indirecte, mais en plus complexe, et n'est pas traité dans ce cours).

6.4.1 Mode d'adressage direct

Dans ce mode d'adressage, le « mécanisme » faisant le lien entre l'adresse physique et l'adresse logique d'un n-uplet n'est mis en place que si c'est nécessaire (*i.e.* seulement si le n-uplet doit être « déplacé hors de sa page p d'origine) : il s'agit d'un **pointeur de suivi**.



Définition : « pointeur de suivi »

Un **pointeur de suivi** n'est mis en place, et donc associé à un n-uplet, QUE lorsque ce n-uplet ne figure pas (plus) dans sa page d'origine p (et, donc, que son adresse physique est passée de (p, i) à (p', i')). Quand il est mis en place, un pointeur de suivi encapsule l'adresse physique réelle (p', i') du n-uplet auquel il est associé.

Avec ce mécanisme :

- Quand un n-uplet figure dans sa page d'origine p , son adresse physique et son adresse logique sont identiques : il s'agit de l'adresse (p, i) du n-uplet. Il n'y a donc pas dans ce cas de « lien » à faire entre ces 2 adresses étant donné qu'elles sont identiques (il n'y a donc aucun pointeur de suivi associé à ce n-uplet),
- Quand un n-uplet ne figure pas dans sa page d'origine p , son adresse physique (p', i') est différente de son adresse logique, inchangée donc, (p, i) . Dans ce cas, un pointeur de suivi « remplace » le n-uplet à son adresse physique d'origine (p, i) et encapsule l'adresse physique actuelle (p', i') du n-uplet. C'est ce pointeur de suivi qui fait le lien entre les 2 adresses.



Remarque

Si un n-uplet a été déplacé de sa page d'origine p , le pointeur de suivi qui lui est associé est toujours placé dans cette page p .

Quelques précisions s'imposent :

- Un pointeur de suivi est lui aussi identifié par une (et une seule) adresse, celle-ci étant elle aussi de la forme (p, i) : en l'occurrence, il est identifié par l'adresse logique du n-uplet associé qu'il « remplace », celle-ci étant aussi l'adresse physique du pointeur de suivi,
- Du coup, dans une page p , un pointeur de suivi est, tout comme l'est un n-uplet, associé à une case d'indice i du répertoire des déplacements de cette page : cette case contient le déplacement (relatif ou absolu) à faire pour atteindre le 1^{er} octet des données du pointeur de suivi au sein de la page p ,
- Vous l'aurez compris à la lecture des 2 points précédents, les pointeurs de suivi sont mélangés aux n-uplets dans les pages de stockage des n-uplets. **Il n'y a donc pas des pages dédiées au stockage des n-uplets et d'autres pages dédiées au stockage des pointeurs de suivi !**

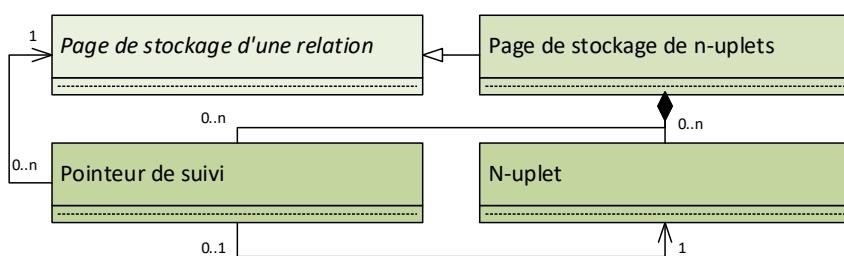


Figure 97. Pages de stockage d'une relation en mode d'adressage direct

Quand on est en mode d'adressage direct, aller « chercher » un n-uplet connu par son adresse logique (p, i) revient donc à aller chercher ce qu'on pense être ce n-uplet. Si c'est bien le n-uplet, pas de souci. Si c'est un pointeur de suivi, on l'exploite en « le suivant » (cf. Figure 98), puisqu'il contient l'adresse physique du n-uplet auquel il est associé.

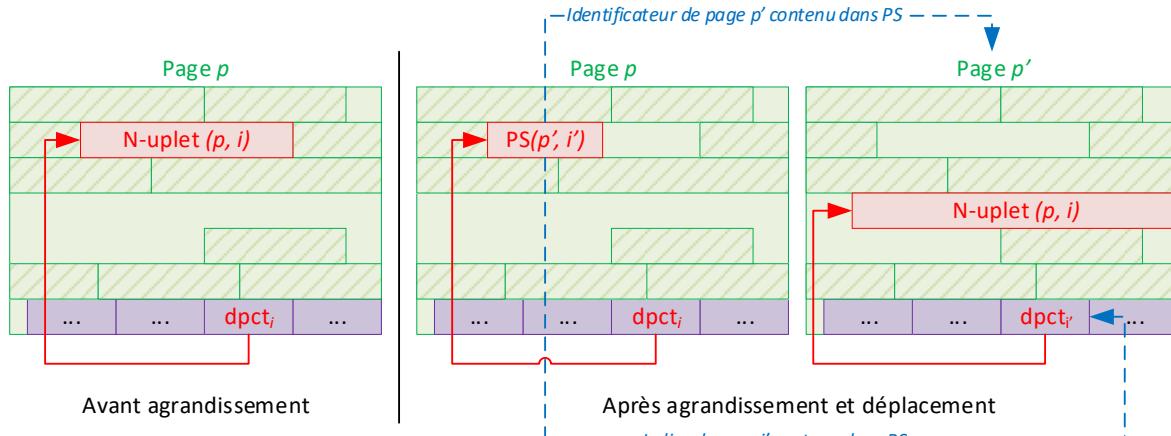


Figure 98. Mise en œuvre de pointeurs de suivi en mode d'adressage direct

En pratique : lecture des données du n-uplet d'adresse logique (p, i)

En mode d'adressage direct, cette lecture peut se faire par le pseudo-algorithme suivant :

Pseudo-algorithme LectureNUAdressageDirect

Données d'entrée

- p : l'identifiant de la page du n-uplet
- i : l'indice de la case associé au n-uplet

Données de sortie

- donnéesNU : les données du n-uplet

Données intermédiaires

- $dpct$: un déplacement
- p' : l'identifiant d'une page
- i' : l'indice d'une case dans le répertoire de p'

Début

- Transférer en mémoire de travail la page p (**1TP_{lecture}**)
- $dpct \leftarrow$ déplacement contenu dans la case d'indice i
- Réaliser le déplacement $dpct$ dans p
- Si** l'on est sur le stockage d'un n-uplet **alors**

 - $\text{donnéesNU} \leftarrow$ lecture des données

- Sinon** //On est sur un pointeur de suivi PS

 - $p' \leftarrow$ identifiant de page contenu dans PS
 - $i' \leftarrow$ indice de case de répertoire contenu dans PS
 - Transférer p' en mémoire de travail (**1TP_{lecture}**)
 - $dpct \leftarrow$ déplacement contenu dans la case i'
 - Réaliser le déplacement $dpct$ dans p'
 - $\text{donnéesNU} \leftarrow$ lecture des données

- FinSi**
- Fin**



Au niveau des performances de la lecture des données d'un n-uplet :

- Si le n-uplet est dans sa page d'origine, cette lecture coûte 1 transfert de page en lecture,
- Sinon, cette lecture coûte 2 transferts de page en lecture : celui de la page contenant le pointeur de suivi et celui de la page contenant ce n-uplet.

Il reste encore un point en suspens...

Question

Lorsqu'on arrive sur la zone de stockage initiale du n-uplet cherché, comment savoir si les données qu'on lit sont celles d'un n-uplet ou celles d'un pointeur de suivi ?



Réponse

Tout simplement en débutant le stockage de ces données par un drapeau (généralement codé sur 1 seul octet, c'est amplement suffisant). Selon sa valeur, on sait comment interpréter les données qui le suivent (*i.e.* comme celles d'un n-uplet ou celles d'un pointeur de suivi). **En mode d'adressage direct, ce drapeau fait toujours parties des métadonnées des n-uplets et des métadonnées des pointeurs de suivi !**

Ainsi, les données que l'on trouve dans un pointeur de suivi sont la valeur de son drapeau suivi de l'adresse physique réelle (p' , i') du n-uplet qui lui est associé.

Remarques

Comme pour les n-uplets, on peut considérer que les données d'un pointeur de suivi sont formées :

- *De données brutes* : il s'agit de l'adresse physique du n-uplet associé,
- *De métadonnées* : le drapeau et éventuellement la case du répertoire qui contient le déplacement à faire pour atteindre les données pointeur de suivi (selon la stratégie de gestion du répertoire des déplacements, cf. §6.6).

Du coup, on a⁸⁶ :

$$T_{totalPS}^{fixe}(R) = T_{donneesPS}^{fixe}(*) + T_{metaPS}^{fixe}(R)$$

Équation 6. Taille totale d'un pointeur de suivi



Et, puisque les données d'un pointeur de suivi sont en fait l'adresse physique du n-uplet auquel il est associé, on a aussi⁸⁶ :

$$T_{donneesPS}^{fixe}(*) = T_{idPage}^{fixe}(*) + T_{indCaseDepl}^{fixe}(*)$$

Équation 7. Taille (fixe) des données (brutes) d'un pointeur de suivi

La taille des métadonnées d'un pointeur de suivi dépend, elle, de la stratégie choisie pour la gestion du répertoire des déplacements (cf. §6.6). Le drapeau (indiquant si on est sur un n-uplet ou sur un pointeur de suivi) fait cependant TOUJOURS partie des métadonnées d'un pointeur de suivi.

⁸⁶ Le « * » indique que cette donnée ne change pas selon la relation considérée : elle est valable pour toute la BD, voire pour tout le SGBD !

Question

Lorsque le n-uplet, d'adresse physique initiale (p, i) , est déplacé de sa page d'origine p vers une nouvelle page p' , il est remplacé par un pointeur de suivi PS , lui-même d'adresse physique (p, i) et contenant sa nouvelle adresse physique (p', i') , c'est entendu... Mais si le n-uplet est de nouveau déplacé vers une nouvelle page p'' , que se passe-t-il ? Y'a-t-il un nouveau pointeur de suivi PS' qui vient le remplacer dans p' et qui va contenir sa nouvelle adresse physique (p'', i'') ? Autrement dit, peut-on avoir un pointeur de suivi vers un pointeur de suivi vers ... vers un n-uplet ?



Réponse

Non ! À tout instant, un n-uplet ne peut être associé qu'à un pointeur de suivi au plus. Du coup, s'il change à nouveau de page pour passer de p' à p'' , on n'introduit pas de nouveau pointeur de suivi : on change l'adresse physique contenue dans le pointeur de suivi PS en la remplaçant par (p'', i'') . Donc, même si le n-uplet a changé 36 fois de pages, sa lecture se fait toujours en 1 ou 2 transferts de pages :

- 1 transfert de page s'il se trouve dans sa page d'origine p ,
- 2 transferts de page sinon : celui de sa page d'origine p afin de récupérer le pointeur de suivi qui lui est associé puis celui de la page dans laquelle il se trouve en réalité.

Question

Et si le n-uplet rétrécit de nouveau ?



Réponse

S'il peut réintégrer sa page d'origine, c'est chose faite. Le pointeur de suivi est alors supprimé et le n-uplet reprend sa place.

6.4.2 Mode d'adressage indirect

Dans ce mode d'adressage, le « mécanisme » faisant le lien entre adresse physique et adresse logique est une **table de correspondance**.



Définition : « table de correspondance »

Une **table de correspondance** associe, pour chaque n-uplet d'une relation R , son adresse physique à son adresse logique (on parle d'ailleurs dans ce cas plutôt « d'identifiant logique »). Comme son nom le suggère, c'est un tableau qui contient :

- Autant de lignes que de n-uplets dans la BD,
- 2 colonnes : une pour les adresses physiques des n-uplets et une pour les identifiants logiques associés.

Adresse physique (variable)	Identifiant logique (invariable)
(p, i)	$idLog_1$
(p', i')	$idLog_2$
(p'', i'')	$idLog_3$
...	...

Figure 99. Contenu d'une table de correspondance

Le principe est donc le suivant :

- Quand un n-uplet est créé, une entrée (une ligne) est ajoutée dans la table de correspondance de la relation R à laquelle appartient ce n-uplet. Cette ligne contient l'adresse physique (p, i) du n-uplet à sa création et un identifiant logique $idLog$ (affecté arbitrairement à ce n-uplet) par lequel le n-uplet est désigné toute sa vie.
- Si le n-uplet change de page, donc si son adresse physique devient (p', i') , il suffit de changer l'information correspondante dans la table de correspondance de la relation R (l'identifiant logique $idLog$ associé à ce n-uplet, lui, restant inchangé).

Ainsi, accéder à un n-uplet en mode d'adressage indirect se fait comme suit (on connaît son identifiant logique $idLog$) : l'entrée de la table de correspondance associée à l'identifiant logique $idLog$ est cherchée et fournit l'adresse physique réelle du n-uplet : il suffit alors d'y accéder « normalement ».

En pratique : lecture des données du n-uplet d'adresse logique $idLog$

La mise en œuvre pratique peut être assimilée au pseudo-algorithme suivant :

Pseudo-algorithme LectureNUAdressageIndirect
Données d'entrée
$idLog$: l'identifiant logique du n-uplet
$TC(R)$: la table de correspondance de la relation R
Données de sortie
$donnéesNU$: les données du n-uplet
Données intermédiaires
$dpct$: un déplacement
p : l'identifiant d'une page
i : l'indice d'une case dans le répertoire de p'
Début
Chercher l'entrée $idLog$ dans $TC(R)$ (???TP_{lecture})
Si elle existe alors
$p \leftarrow$ identificateur de page associé à $idLog$ dans $TC(R)$
$i \leftarrow$ indice de case associé à $idLog$ dans $TC(R)$
Transférer p en mémoire de travail (1TP_{lecture})
$dpct \leftarrow$ déplacement contenu dans la case i
Réaliser le déplacement $dpct$ dans p
$donnéesNU \leftarrow$ lecture des données
Sinon
Signaler une erreur « données inexistantes »
FinSi
Fin



Le point « incertain » dans cette méthode est le coût de la recherche au sein de la table de correspondance (et donc d'écriture si un n-uplet est déplacé d'une page vers une autre). Pour aborder cet aspect, il est nécessaire de discuter du stockage d'une table de correspondance (cf. Figure 101) : comme tout, une table de correspondance est stockée au sein de pages qui ont une structure propre : **elles ne contiennent QUE des lignes de la forme $((p, i), id)$.**



Remarque

Il est intéressant de noter que :

- Une table de correspondance contient autant de lignes qu'il y a de n-uplets dans la relation R à laquelle elle est associée :

$$nb_{lignesTC}^{exact}(R) = nb_{n-uplets}^{exact}(R) = Card(R)$$

Équation 8. Nombre de lignes dans une table de correspondance

- Toutes lignes d'une même table de correspondance ont la même taille (les tailles des différentes valeurs de p étant fixes, idem pour les valeurs de i et celles de $idLog$)⁸⁷ :

$$T_{ligneTC}^{fixe}(*) = \left(\underbrace{T_{idPage}^{fixe}(*) + T_{idCaseDepl}^{fixe}(*)}_{\text{adresse physique du n-uplet}} \right) + \underbrace{T_{idLog}^{fixe}(*)}_{\text{adresse logique du n-uplet}}$$

Équation 9. Taille (fixe) d'une ligne d'une table de correspondance

Dans ce mode d'adressage, le mécanisme faisant le lien entre adresse logique et adresse physique d'un n-uplet (*i.e.* la table de correspondance) n'est pas stocké au sein des mêmes pages que celles qui contiennent les n-uplets⁸⁸ ! En mode d'adressage indirect, une relation est donc stockée sur 2 « types » de pages : les pages de stockage des n-uplets d'une part et les pages de stockages des entrées de la table de correspondance de la relation d'autre part.

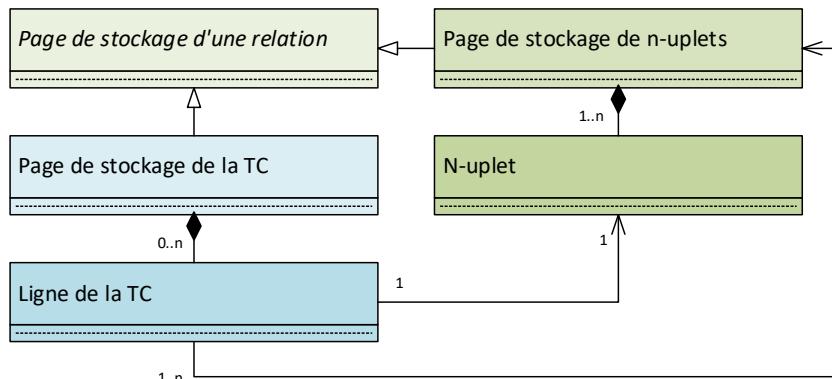


Figure 100. Pages de stockage d'une relation en mode d'adressage indirect

⁸⁷ Le « * » indique que cette donnée ne change pas selon la relation considérée : elle est valable pour toute la BD, voire pour tout le SGBD !

⁸⁸ Contrairement, donc, au mécanisme faisant ce même lien en mode d'adressage direct (cf. §6.4.1) : les pointeurs de suivi sont en effet stockés dans les mêmes pages que les n-uplets auxquels ils sont associés.

Ces 2 types de pages (celles qui contiennent les n-uplets de la relation R et celles qui contiennent les lignes de la table de correspondance associée à la relation R) « s'organisent » comme suit :



Remarques

De plus, toujours par souci de simplicité de gestion, **une entrée d'une table de correspondance n'est JAMAIS stockée à cheval sur plusieurs pages : elle est toujours complètement écrite dans une et une seule page** (et la question d'éventuelles « entrées longues » ne se pose pas : ça n'existe pas !).

Étant donné que toutes les lignes d'une table de correspondance ont la même taille (cf. Équation 9), aucun mécanisme particulier (genre « répertoire des déplacements » ou autre) n'est nécessaire pour pouvoir atteindre une ligne de la table de correspondance au sein des pages dans lesquelles ces lignes sont stockées : on sait toujours où débute, dans chacune de ces pages, chaque ligne de la table de correspondance qu'elle contient.

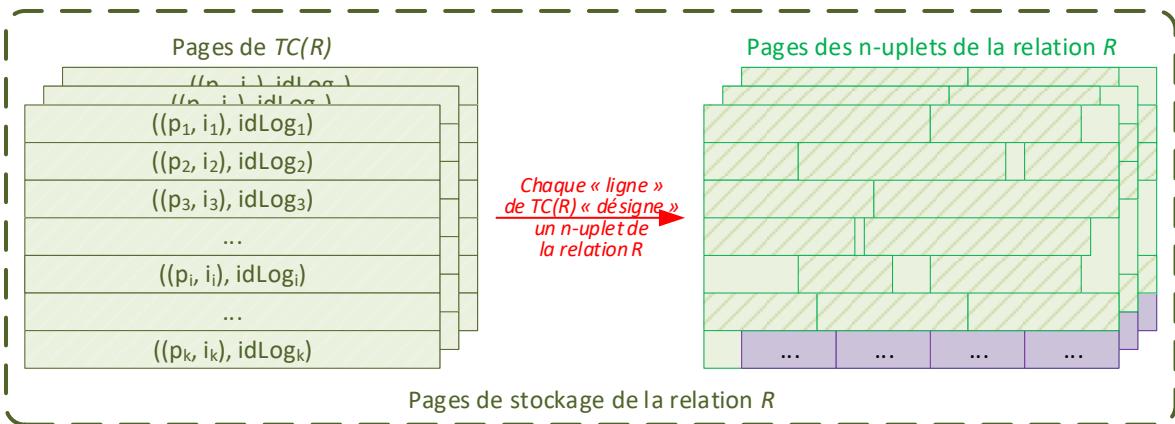


Figure 101. Pages d'une table de correspondance vs pages de n-uplets pour une relation R

Partant de là, pour rebondir sur le coût du pseudo-algorithme de lecture des données du n-uplet d'adresse logique $idLog^{89}$, il est possible de calculer la taille, en nombre de pages, occupée par une table de correspondance d'une relation R (puisque l'on sait qu'elle est stockée dans des pages spécifiques ne contenant que des lignes de cette table de correspondance et qu'on connaît le nombre et la taille, fixe, de ces lignes). On peut donc avoir une idée du nombre de transferts de pages à faire en lecture (au sein des pages dans lesquelles est stockée la table de correspondance) pour trouver dans $TC(R)$ une entrée associée à l'identifiant logique $idLog$ recherché. Il faut penser à ajouter 1 à ce nombre : une fois qu'on a l'entrée souhaitée dans $TC(R)$, il est nécessaire de transférer en lecture la page p (cette fois une page de stockage de n-uplets) contenant effectivement le n-uplet cherché, pour aller lire ses données.



Remarque

Usuellement (assez souvent), les tables de correspondances sont intégralement chargées au lancement du SGBD dans une zone dédiée de la mémoire de travail⁹⁰. Leur exploitation (pour chercher une ligne associée à un identifiant logique $idLog$) se fait donc à coût nul si c'est le cas.

⁸⁹ Cf. pseudo-algorithme LectureNUAdressageIndirect vu plus haut.

⁹⁰ Un peu comme tout ou partie des tables système.

6.4.3 Mode d'adressage direct vs indirect

Vous l'aurez compris, chacun de ces modes d'adressage présente des avantages et des inconvénients...

Stratégies d'adressage des n-uplets	Avantages	Inconvénients
Mode d'adressage direct	Mécanisme de lien entre adresses logiques et adresses physiques mis en place uniquement lorsque c'est nécessaire, la place utilisée par ce mécanisme est donc réduite.	Ajout d'un nouveau type « d'objet » (<i>i.e.</i> les pointeurs de suivis) dans les pages de stockage des n-uplets, d'où un accroissement de la complexité de gestion de ces pages (mise en place de drapeaux nécessaire).
Mode d'adressage indirect	Simplicité de gestion : le mécanisme de lien entre adresses logiques et adresses physiques (<i>i.e.</i> la table de correspondance) est gérée à part (<i>i.e.</i> dans d'autres pages) que les n-uplets.	Table de correspondance mise en place en permanence pour chaque relation, d'où une éventuelle place « perdue ».

Tableau 25. Bilan des stratégies d'adressage des n-uplets

6.5 Stratégies de stockage des valeurs d'attributs

Comme nous l'avons vu, les données brutes d'un n-uplet sont en fait constituées des valeurs des attributs pour ce n-uplet. Stocker de telles valeurs ne consiste pas simplement à aligner une suite de 0 et de 1 (codage binaire des valeurs). En effet, une fois que l'on atteint le début de la zone de stockage d'un n-uplet, si on a besoin de la valeur de son 4^{ème} attribut, il est nécessaire de savoir à quel octet commence cette valeur-là et combien d'octets il est nécessaire de lire pour pouvoir la récupérer !

Il est donc nécessaire de choisir une stratégie de stockage de ces valeurs permettant, comme toujours, d'allier au maximum performances (ici en minimisant la place « perdue ») et simplicité de gestion. Le choix d'un format doit prendre en compte la possibilité de représenter des attributs dont les valeurs (en fait, leur codage binaire) n'ont pas toutes la même longueur, l'évolutivité des schémas de relation par ajout ou suppression d'attributs et la rapidité d'accès aux valeurs d'attributs. Ainsi, 2 grandes stratégies « s'affrontent » :

- *Le stockage des valeurs d'attributs en format fixe* : cette stratégie impose de réservé, pour toute valeur d'attribut, un nombre d'octets égal au nombre d'octets que peut occuper au maximum cette valeur (cette taille maximum est obtenue en fonction du type de l'attribut considéré⁹¹) ; ainsi, quand on est sur une valeur d'attribut, en « regardant » son type on sait directement sur combien d'octets est stockée cette valeur et, donc, où débute la valeur de l'attribut suivant.



Remarque

Implicitement, avec cette stratégie de stockage des valeurs d'attributs, tous les n-uplets d'une même relation font la même taille⁹².

⁹¹ Certains types spécifiques, notamment les « variants », imposent la mise en place d'autres mécanismes. Nous ne traiterons pas ces types particuliers ici.

⁹² Sauf, encore une fois, cas particulier des n-uplets dont un ou plusieurs attributs sont typés grâce à des types spécifiques tels que les « variants » (types non traités ici).

- *Le stockage des valeurs d'attributs en format variable* : chaque valeur d'attribut est codée exactement sur le nombre d'octets nécessaire à son codage ; en revanche, le SGBD doit aussi stocker d'autres informations en précédant chaque valeur d'un attribut de sa taille réelle⁹³. **Ces tailles des valeurs d'attributs sont donc à compter dans la taille des métadonnées du n-uplet (la taille indiquée pour une valeur d'un attribut occupant en effet elle-même de la place).**



Exemple

Imaginons que l'on souhaite coder la valeur 427 comme une valeur de type unsigned int. Imaginons également que la plus grande valeur de type unsigned int nécessite 4 octets pour être codée. Ainsi :

- *En stockage de valeurs d'attributs en format fixe* :

Valeur de l'attribut			
Octet PF	Octet Pf
0000 0000	0000 0000	0000 0001	1010 1011

- *En stockage des valeurs d'attributs en format variable*, en supposant que l'on code la taille de la valeur de chaque attribut sur 1 octet :

Taille de la valeur	Valeur de l'attribut	
	Octet PF	Octet Pf
0000 0010	0000 0001	1010 1011

Encore une fois, ces stratégies ont chacune leurs avantages et leurs inconvénients...

Stratégies de stockage des valeurs d'attributs	Avantages	Inconvénients
En format fixe	Simplicité de gestion	Place souvent perdue pour le stockage des valeurs d'attributs
En format variable	Minimisation fréquente de la place perdue pour le stockage des valeurs d'attributs	Place occupée par les métadonnées nécessaires (taille de la valeur effective de chaque attribut) et gestion de ces métadonnées

Tableau 26. Bilan des stratégies de stockage des valeurs d'attributs



Remarque

Souvent, toujours dans l'optique de simplifier la gestion, **si on est « en format variable », on considère que toutes les tailles de tous les attributs de tous les n-uplets d'une même BD⁹⁴ sont codées sur le même nombre d'octets.**

Du coup, pour être complet, un point reste en suspens dans le cadre du stockage des valeurs d'attributs en format variable...

⁹³ plus rarement du nom de l'attribut, précédé d'un code spécial.

⁹⁴ Voir de toutes les BD gérées par le même SGBD.

Question

Sur combien d'octets coder la taille de la valeur d'un attribut ?



Réponse

Nous venons de le voir, si on est « en format variable », toutes les tailles de tous les attributs de tous les n-uplets d'une même BD⁹⁴ sont codées sur le même nombre d'octets. Il est donc important de choisir le nombre d'octets occupé par les tailles des valeurs d'attributs de façon à ne pas perdre trop de place (dire qu'elles sont toutes codées sur 20 octets est réellement inutile : regardez la taille que pourraient faire toutes les valeurs de tous les attributs !) mais aussi de façon à ne pas trop restreindre la place que pourrait prendre une valeur d'un attribut basé sur un type permettant de grandes valeurs (ou alors cela peut-être une stratégie pour limiter la place occupée par les valeurs d'attributs, même pour celles qui sont basées sur un type de données « large »). À vous de voir, donc...

6.6 Stratégies de gestion des répertoires des déplacements

Nous l'avons vu (cf. §6.1), chaque n-uplet stocké dans une page est associé à une case du répertoire des déplacements situés à la fin de cette même page. Cela amène une question...



Questions

Puisque chaque n-uplet situé dans une page p est associé à une case i du répertoire des déplacements de cette page p , cela signifie que si le répertoire des déplacements contient k cases, alors on ne pourra pas placer dans la page p plus de k n-uplets, et ce même s'ils pourraient « tenir » dedans ? À l'inverse, si la page ne contient qu'un ou deux n-uplets, y'a-t-il des cases « inutilisées » dans le répertoire des déplacements située à sa fin ?

Réponse

Oui et non : tout dépend de la stratégie choisie pour la gestion du répertoire des déplacements.

Il existe plusieurs façons de gérer le répertoire des déplacements situé dans une page de stockage des n-uplets. Les plus courantes sont :

- *La stratégie de gestion statique du répertoire des déplacements* : l'idée est de prévoir dès la création d'une page de stockage de n-uplets un nombre de cases dans le répertoire des déplacements suffisant pour pouvoir stocker un nombre « raisonnable » de n-uplets dans la page (la taille du répertoire des déplacements est donc fixe),
- *La stratégie de gestion dynamique du répertoire des déplacements* : on ajoute au répertoire des déplacements une case à chaque fois que l'on ajoute un n-uplet dans la page et on en supprime une⁹⁵ lorsque l'on supprime un n-uplet de la page.



Attention

Supprimer une case du répertoire des déplacements ou le défragmenter provoque usuellement le changement de l'indice i de la case du répertoire des déplacements associée à des n-uplets contenus dans la page. Ces changements d'indices changent l'adresse physique de ces n-uplets : il faut donc en tenir compte (selon le mode d'adressage choisi, cf. §6.4) !

⁹⁵ Parfois avec un travail similaire à de la défragmentation, mais cela pose d'autres problèmes.

Comme d'habitude, chaque stratégie a ses avantages et ses inconvénients...

Stratégies de gestion du répertoire des déplacements	Avantages	Inconvénients
Statique	Simplicité de gestion	Possible perte de place (s'il y a peu de n-uplets dans la page, des cases du répertoire des déplacements sont inutilisées ou, au contraire, si toutes les cases du répertoire des déplacements sont prises, les données des n-uplets n'occupent peut-être au mieux la zone de stockage)
Dynamique	Perte de place limitée le plus possible	Gestion plus coûteuse (donc baisse de performances) Éventuelle nécessité de défragmentation Éventuelle nécessité de mécanismes complémentaires selon la stratégie d'adressage retenue.

Tableau 27. Bilan des stratégies de gestion du répertoire des déplacements

Il est également nécessaire de bien dimensionner la taille des cases de ce répertoire des déplacements. En effet, chacune de ces cases contient un déplacement, c'est-à-dire ici un nombre entier positif ou nul d'octets dont il faut se déplacer (soit de façon absolue depuis le début de la page, soit de façon relative depuis la case considérée dans le répertoire des déplacements de la page) pour « trouver » le premier octet de la zone de stockage des données du n-uplet associé à la case. Il ne sert donc à rien de prévoir une place trop grande dans chaque case du répertoire des déplacements (une partie de cette place serait alors perdue puisqu'inutilisée) ni de prévoir une place trop petite (il serait alors impossible d'atteindre tous les octets de la zone de stockage des n-uplets).



Exemple

Admettons que la taille des pages soit de 2 048 octets. On sait donc que, au maximum, la taille de la zone de stockage des n-uplets dans cette page sera d'un peu moins de 2 048 octets (la taille de la page – la taille du répertoire des déplacements qui sera implanté à la fin de cette page). Ainsi :

- Si on fixe la taille des cases du répertoire des déplacements à 1 octet, on pourra stocker dans ces cases des valeurs de déplacements comprises entre 0 et 255 octets. Que le déplacement soit relatif ou absolu, on sait qu'on ne pourra alors pas atteindre la totalité de la zone de stockage des n-uplets dans cette page.
- Si on fixe la taille des cases du répertoire des déplacements à 4 octets, on pourra stocker dans ces cases des valeurs de déplacements comprises entre 0 et 4 294 967 295 octets. Que le déplacement soit relatif ou absolu, pouvoir stocker des déplacements aussi grands est ici inutile.

Avec une taille de pages de 2 048 octets, une bonne taille de cases pour les répertoires des déplacements est de 2 octets : on peut alors stocker dans ces cases des valeurs de déplacements comprises entre 0 et 65 535 octets (cette valeur de 65 535 étant la plus petite puissance de 2 immédiatement supérieure ou égale à la taille des pages qui est ici de 2 048 octets).



Remarque

Dans l'optique de simplifier la gestion des données par le SGBD, tout en optimisant son fonctionnement, on uniformise forcément certaines choses. Ainsi :

- La stratégie choisie pour la gestion d'un répertoire (statique ou dynamique) est la même pour toutes les pages de stockage des n-uplets d'une BD⁹⁶,
- La taille des cases des répertoires situés à la fin des pages de stockage des n-uplets d'une BD est la même pour toutes ces pages⁹⁷.

Enfin, si la stratégie de gestion du répertoire des déplacements choisie est la gestion dynamique, on peut observer que chaque n-plet contenu dans une page est associé à une et une seule case du répertoire des déplacements de cette page et que chaque case du répertoire des déplacements d'une page est associée à un et un seul n-plet de cette-même page : il y a donc toujours égalité entre le nombre de n-uplets contenus dans une page et le nombre de cases contenues dans le répertoire des déplacements de cette page. Ainsi, assez logiquement, **en cas de stratégie de gestion dynamique du répertoire des déplacements, on peut considérer qu'une case du répertoire est une métadonnée d'un n-plet.**



Question

Et en cas de stratégie de gestion statique du répertoire des déplacements, comment sont prises en compte les cases de ce répertoire ?

Réponse

Elles sont prises en compte en retranchant de la taille de la page la place occupée par le répertoire des déplacements tout entier : puisqu'il est de taille fixe, on va décompter sa taille de celle de la page afin de savoir quelle place il reste dans la page pour y stocker des n-uplets.

Enfin, le choix d'une stratégie de gestion du répertoire des déplacements à un impact sur les métadonnées des n-uplets :

- Avec une gestion dynamique du répertoire des déplacements : on a une bijection entre les n-uplets/éventuels pointeurs de suivi⁹⁸ contenus dans une page et les cases du répertoire des déplacements de cette même page. **Dans ce cas, la place occupée par une case du répertoire des déplacements doit être comptée dans la place occupée par les métadonnées d'un n-plet/éventuel pointeur de suivi⁹⁸.**
- Avec une gestion statique du répertoire des déplacements : le nombre de cases contenues dans ce répertoire des déplacements est fixe et indépendant du nombre de n-uplets contenus dans la page. Les cases du répertoire des déplacements ne sont donc pas assimilables à des métadonnées des n-uplets/éventuels pointeurs de suivi⁹⁸ ! En revanche, il convient de retrancher de la taille de la page la place (fixe) occupée par le répertoire des déplacements pour savoir quelle place reste disponible pour stocker des n-uplets/éventuels pointeurs de suivi⁹⁸ dans cette page.

⁹⁶ Voir pour toutes les BD gérées par un même SGBD.

⁹⁷ Voir même pour toutes les pages de stockage des n-uplets de toutes les BD gérées par un même SGBD.

⁹⁸ Si on est en mode d'adressage direct.

6.7 Impact des différentes stratégies au niveau des performances

On l'a vu (cf. §5.1.3), améliorer les performances passe notamment par la minimisation des pages occupées (moins on a de pages, moins on aura *a priori* de pages à transférer !). Sont ici résumés les impacts des différents choix stratégiques au niveau de la place occupée par une relation...

On a tout de même une donnée constante :

$$T_{totaleNU}^{fixe|moy|min|max}(R) = T_{donneesNU}^{fixe|moy|min|max}(R) + T_{metaNU}^{fixe}(R)$$

Équation 10. Taille (totale) d'un n-uplet d'une relation R

6.7.1 Impact du choix du mode d'adressage

Le mode d'adressage direct implique la prise en compte d'éventuels pointeurs de suivi alors que le mode d'adressage indirect implique la prise en compte d'une table de correspondance. Ainsi :

- *En mode d'adressage direct :*
 - Un drapeau (qui occupe une certaine place) doit être compté parmi les métadonnées des n-uplets⁹⁹ :

$$T_{metaNU}^{fixe}(R) = T_{drapeauNUPS}^{fixe} (*) + \underbrace{\dots}_{\text{autres métadonnées}}$$

Équation 11. Adressage direct : impact sur les métadonnées des n-uplets

- Les pointeurs de suivi doivent aussi être comptabilisés⁹⁹ :

$$T_{totalePS}^{fixe}(R) = T_{donneesPS}^{fixe} (*) + T_{metaPS}^{fixe}(R)$$

Équation 12. Adressage direct : taille (totale) d'un pointeur de suivi

$$T_{donneesPS}^{fixe} (*) = T_{idPage}^{fixe} (*) + T_{idCaseDepl}^{fixe} (*)$$

Équation 13. Adressage direct : taille des données (brutes) d'un pointeur de suivi

$$T_{metaPS}^{fixe}(R) = T_{drapeauNUPS}^{fixe} (*) + \underbrace{\dots}_{\text{autres métadonnées}}$$

Équation 14. Adressage direct : taille des métadonnées des pointeurs de suivi



Remarque

On ne peut RIEN dire quant au nombre de pointeurs de suivi, si ce n'est le borner de façon évidente :

$$\forall R, 0 \leq nb_{PS}^{exact}(R) \leq Card(R)$$

Équation 15. Bornage (trivial) du nombre de pointeurs de suivi dans une relation R

Cependant, quand des pointeurs de suivi sont présents, on considère très souvent qu'ils sont répartis de façon équiprobable sur les pages de stockages des n-uplets (et des pointeurs de suivi, donc !) de la relation.

⁹⁹ Le « * » indique que cette donnée ne change pas selon la relation considérée : elle est valable pour toute la BD, voire pour tout le SGBD !

- En mode d'adressage indirect : aucun pointeur de suivi n'est mis en œuvre et rien n'est à prendre en compte au niveau des métadonnées des n-uplets. En revanche, il faut tenir compte de la table de correspondance et de son stockage¹⁰⁰ !

$$T_{ligneTC}^{fixe}(*) = \left(\frac{T_{idPage}^{fixe}(*) + T_{idCaseDepl}^{fixe}(*)}{\text{adresse physique du n-uplet}} \right) + \frac{T_{idLog}^{fixe}(*)}{\text{adresse logique du n-uplet}}$$

Équation 16. Taille (fixe) d'une ligne d'une table de correspondance

$$nb_{lignesTC}^{exact}(R) = nb_{n-uplets}^{exact}(R) = Card(R)$$

Équation 17. Nombre de lignes dans une table de correspondance

6.7.2 Impact du choix du format de stockage des valeurs d'attributs

Soit ces valeurs occupent une place fixe, soit elles occupent une taille variable mais il faut prendre en compte le fait que cette taille est alors effectivement stockée elle-aussi (pour chaque attribut) et qu'elle occupe une certaine place :

- En format fixe de stockage des valeurs d'attributs : chaque valeur d'attribut est codée sur la place occupée par la plus grande des valeurs codables sur le type de cet attribut. Cela influe donc sur la taille des données des n-uplets :

$$T_{donneesNU}^{fixe}(R) = \sum_{i=1}^{i \leq nb_{attributs}^{fixe}(R)} T_{valAttr}^{fixe}(R, att_i)$$

Équation 18. Format fixe : taille des données (brutes) des n-uplets

- En format variable de stockage des valeurs d'attributs : chaque valeur est codée sur la place strictement nécessaire pour la coder et est précédée de cette indication de place. Cela influe sur la taille des données des n-uplets mais aussi sur celles de leurs métadonnées¹⁰⁰ :

$$T_{donneesNU}^{moy|min|max}(R) = \sum_{i=1}^{i \leq nb_{attributs}^{fixe}(R)} T_{valAttr}^{moy|min|max}(R, att_i)$$

Équation 19. Format variable : taille des données (brutes) des n-uplets

$$T_{metaNU}^{fixe}(R) = (T_{taillevalAttr}^{fixe}(*) \times nb_{attributs}^{fixe}(R)) + \underbrace{\dots}_{\text{autres métadonnées}}$$

Équation 20. Format variable : impact sur les métadonnées des n-uplets

¹⁰⁰ Le « * » indique que cette donnée ne change pas selon la relation considérée : elle est valable pour toute la BD, voire pour tout le SGBD !

6.7.3 Impact du choix de gestion du répertoire des déplacements

En gestion statique, ce répertoire des déplacements a une taille variable (autant de cases que de n-uplets contenus dans la page) alors qu'en format statique il a une taille fixe.

- *Gestion statique du répertoire des déplacements* : le nombre de cases contenues dans le répertoire des déplacements est décorrélé du nombre de n-uplets (voire de pointeurs de suivi si on est en mode d'adressage direct) contenus dans la même page. On ne peut donc PAS associer ces cases à des métadonnées des n-uplets (et des pointeurs de suivi si on est en mode d'adressage direct). Cependant, on doit décompter globalement la place (fixe, donc) occupée par le répertoire des déplacements de la taille de la page pour savoir quelle place est disponible dans la page pour y stocker des n-uplets (voire des pointeurs de suivi en mode d'adressage direct)¹⁰¹ :

$$T_{utileDsPage}^{fixe}(*) = T_{page}^{fixe}(*) - (nb_{casesDepl}^{fixe}(*) \times T_{caseDepl}^{fixe}(*))$$

Équation 21. Gestion statique : impact sur la place « utile » restant dans une page

- *Gestion dynamique du répertoire des déplacements* : le nombre de cases contenues dans le répertoire des déplacements est corrélé au nombre de n-uplets (voire de pointeurs de suivi si on est en mode d'adressage direct) contenus dans la même page. On DOIT donc associer ces cases à des métadonnées des n-uplets (et des pointeurs de suivi si on est en mode d'adressage direct) :

$$T_{utileDansPage}^{fixe}(R) = T_{page}^{fixe}(*)$$

Équation 22. Gestion dynamique : aucun impact sur la place « utile » dans une page

$$T_{metaNU}^{fixe}(R) = T_{caseDepl}^{fixe}(*) + \underbrace{\dots}_{\substack{\text{autres} \\ \text{métadonnées}}}$$

Équation 23. Gestion dynamique : impact sur les métadonnées des n-uplets

$$T_{metaPS}^{fixe}(R) = T_{caseDepl}^{fixe}(*) + \underbrace{\dots}_{\substack{\text{autres} \\ \text{métadonnées}}}$$

Équation 24. Gestion dynamique (SI adressage direct) : impact sur les métadonnées des pointeurs de suivi

¹⁰¹ Le « * » indique que cette donnée ne change pas selon la relation considérée : elle est valable pour toute la BD, voire pour tout le SGBD !

6.8 Paramètres de l'organisation physique

En résumé, voici les paramètres intervenant au niveau de la gestion des accès aux données :

- *Paramètres généraux du SGBD (indépendants de toute relation R)¹⁰²* :

Notation	Unité(s) usuelle(s)	Paramètre (fourni)
-	-	Stratégie d'adressage des n-uplets (direct ou indirect).
-	-	Stratégie de stockage des valeurs d'attributs (fixe ou variable).
-	-	Stratégie de gestion du répertoire des déplacements situé à la fin des pages de stockage des n-uplets (statique ou dynamique).
$T_{idCaseDepl}^{fixe}(*)$	octets	Taille (fixe) d'un indice d'une case d'un répertoire des déplacements situé à la fin des pages de stockage des n-uplets.
$T_{caseDepl}^{fixe}(*)$	octets	Taille (fixe) d'une case d'un répertoire des déplacements situé à la fin des pages de stockage des n-uplets.
$nb_{casesDepl}^{fixe}(*)$	cases du rép. des dépl.	En gestion statique du répertoire des déplacements uniquement : nombre (fixe) de cases contenues dans le répertoire des déplacements situé à la fin des pages de stockage des n-uplets.
$T_{tailleValAttr}^{fixe}(*)$	octets	En format variable de stockage des valeurs d'attributs uniquement : place (fixe) occupée par l'indication de la taille effective d'une valeur d'attribut.
$T_{caractere}^{fixe}(*)$	octets	Place (fixe) occupée par un caractère (utile si on nous donne la taille des valeurs d'attributs en nombre de caractères et non en octets).
$T_{drapeauNUPS}^{fixe}(*)$	octets	En adressage direct uniquement : taille (fixe) d'un drapeau indiquant si ce qui suit correspond au stockage d'un n-uplet ou à celui d'un pointeur de suivi.
$T_{idLog}^{fixe}(*)$	octets	En mode d'adressage indirect uniquement : taille (fixe) d'un identificateur logique de n-uplet.

Tableau 28. Performances d'accès aux données : paramètres SGBD (génériques)

¹⁰² Le « * » indique que cette donnée ne change pas selon la relation considérée : elle est valable pour toute la BD, voire pour tout le SGBD !

- Paramètres d'une relation R au niveau des pages de stockage de ses n-uplets¹⁰³ :

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$Card(R)$	n-uplets	Cardinalité de (i.e. nombre de n-uplets contenus dans) la relation R : $Card(R) \equiv nb_{NU}^{exact}(R)$
$nb_{attributs}^{fixe}(R)$	attributs	Nombre d'attributs définis dans le modèle (intension) de la relation R.
$T_{totaleNU}^{fixe moy min max}(R)$	octets	Taille totale (fixe ou moyenne ou minimale ou maximale) d'un n-uplet de la relation R : $T_{totaleNU}^{fixe moy min max}(R) = T_{donneesNU}^{fixe moy min max}(R) + T_{metaNU}^{fixe}(R)$
$T_{donneesNU}^{fixe moy min max}(R)$	octets	Taille (fixe ou moyenne ou minimale ou maximale) des données « brutes » d'un n-uplet de la relation R : $T_{donneesNU}^{fixe moy min max}(R) = \sum_{i=1}^{i \leq nb_{attributs}^{fixe}(R)} T_{valAttr}^{fixe moy min max}(R, att_i)$
$T_{valAttr}^{fixe moy min max}(R, att)$	octets ou caractères	Taille (fixe ou moyenne ou minimale ou maximale) des valeurs de l'attribut att dans les n-uplets de la relation R. <div style="background-color: #e0e0e0; padding: 10px; border-radius: 10px;"> Attention Si elle est donnée en caractères, n'oubliez pas de la « convertir » afin de l'obtenir en octets !!! Pour cela, multipliez la taille donnée caractères par la place occupée par chaque caractère (cf. Tableau 28) qui vous sera alors forcément indiquée. </div>
$T_{metaNU}^{fixe}(R)$	octets	Taille (fixe) des métadonnées des n-uplets de la relation R. La nature de ces métadonnées (et donc leur taille) dépend des choix stratégiques faits ici : $T_{metaNU}^{fixe}(R) = \underbrace{T_{drapeauNUPS}^{fixe}(*)}_{si adresse direct} + \underbrace{\left(T_{tailleValAttr}^{fixe}(*) \times nb_{attributs}^{fixe}(R) \right)}_{si format variable} + \underbrace{T_{caseDepl}^{fixe}(*)}_{si gestion dynamique}$

Tableau 29. Performances d'accès aux données : paramètres SGBD (n-uplets d'une relation R)

¹⁰³ Le « * » indique que cette donnée ne change pas selon la relation considérée : elle est valable pour toute la BD, voire pour tout le SGBD !

- Paramètres d'une relation R au niveau de ses pointeurs de suivi ou de sa table de correspondance (l'un OU l'autre, selon le mode d'adressage choisi)¹⁰⁴ :

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$T_{totalePS}^{fixe}(R)$	octets	En mode d'adressage direct uniquement : taille totale (fixe) des pointeurs de suivi : $T_{totalePS}^{fixe}(R) = T_{donneesPS}^{fixe}(*) + T_{metaPS}^{fixe}(R)$
$T_{donneesPS}^{fixe}(*)$	octets	En mode d'adressage direct uniquement : taille (fixe) des données « brutes » d'un pointeur de suivi : $T_{donneesPS}^{fixe}(*) = T_{idPage}^{fixe}(*) + T_{idCaseDepl}^{fixe}(*)$
$T_{metaPS}^{fixe}(R)$	octets	En mode d'adressage direct uniquement : taille (fixe) des métadonnées d'un pointeur de suivi : $T_{metaPS}^{fixe}(R) = T_{drapeauNUPS}^{fixe}(*) + \underbrace{T_{caseDepl}^{fixe}(*)}_{\text{si gestion dynamique}}$
$nb_{PS}^{fixe moy min max}(R)$	pointeurs de suivi	En mode d'adressage direct uniquement : nombre total (fixe ou moyen ou minimal ou maximal) de pointeurs de suivi stockés avec les n-uplets de la relation R. On a forcément le bornage suivant : $0 \leq nb_{PS}^{fixe moy min max}(R) \leq Card(R)$
$nb_{PS/NU}^{fixe moy min max}(R)$	pointeurs de suivi par n-uplet	En mode d'adressage direct uniquement : nombre (fixe ou moyen ou minimal ou maximal) de pointeurs de suivi par n-uplet de la relation R ¹⁰⁵ : $nb_{PS/NU}^{fixe moy min max}(R) \xrightarrow{\text{ou}} \frac{Card(R)}{nb_{PS}^{fixe moy min max}(R)}$
$T_{ligneTC}^{fixe}(*)$	octets	En mode d'adressage indirect uniquement : taille (fixe) d'une ligne de la table de correspondance de la relation R : $T_{ligneTC}^{fixe}(*) = (T_{idPage}^{fixe}(*) + T_{idCaseDepl}^{fixe}(*)) + T_{idLog}^{fixe}(*)$
$nb_{lignesTC}^{exact}(R)$	lignes de la TC	En mode d'adressage indirect uniquement : nombre de lignes contenues dans la table de correspondance de la relation R : $nb_{lignesTC}^{exact}(R) = Card(R)$

Tableau 30. Performances d'accès aux données : paramètres SGBD (PS ou TC d'une relation R)

¹⁰⁴ Le « * » indique que cette donnée ne change pas selon la relation considérée : elle est valable pour toute la BD, voire pour tout le SGBD !

¹⁰⁵ Il faudra « faire des tests » avec l'arrondi par défaut ET avec l'arrondi par excès.

Vous avez maintenant tout pour calculer la place occupée par une relation (tout compris) !



Attention

N'oubliez pas que, sauf les n-uplets/attributs longs, RIEN ne peut être stocké à cheval sur 2 pages (ni n-uplets, ni pointeurs de suivi, ni lignes d'une table de correspondance si on est en mode d'adressage indirect) ! **Tous les calculs DOIVENT donc être faits localement (par « type » de page) et non globalement !!!**

N'oubliez pas non plus que, en cas de mode d'adressage direct, n-uplets et pointeurs de suivi sont mélangés au sein des mêmes pages !

Puisque les calculs doivent se faire localement (par « type » de page), on va d'abord calculer combien « d'informations » on peut mettre au mieux dans chaque « type » de page : les pages de stockage des n-uplets contiennent (en plus du répertoire des déplacements) des n-uplets (*sic*) et, éventuellement, des pointeurs de suivi (pour ces derniers, si on est en mode d'adressage direct uniquement) ; les pages de stockage de la table de correspondance ne contiennent que des lignes de cette table. On a donc¹⁰⁶ :

Notation	Unité(s) usuelle(s)	Paramètre (calculé)
$T_{utileDsPageNU}^{fixe}(*)$	octets	Taille (fixe) « utile » dans une page de stockage des déplacements (<i>i.e.</i> place utilisable dans la page pour y stocker des n-uplets et d'éventuels pointeurs de suivi) : $T_{utileDsPageNU}^{fixe}(*) = T_{page}^{fixe}(*) - \underbrace{(nb_{casesDepl}^{fixe}(*) \times T_{caseDepl}^{fixe}(*))}_{\text{si gestion statique uniquement}}$
$nbMax_{NU/page}^{moy max}(R)$	n-uplets par page	Nombre (moyen ou maximal) de n-uplets au plus par page : $\frac{T_{utileDsPageNU}^{fixe}(*)}{T_{NU}^{fixe moy min max}(R) + \underbrace{(nb_{PS/NU}^{fixe moy min max}(R) \times T_{PS}^{fixe}(R))}_{=0 \text{ si adressage indirect}}}$
$nbMax_{PS/page}^{moy max}(R)$	pointeurs de suivi par page	En mode d'adressage direct uniquement : nombre (moyen ou maximal) de pointeurs de suivi au plus par page ¹⁰⁷ : $nbMax_{PS/page}^{moy max}(R) \xrightarrow{\text{ou}} nb_{NU/page}^{moy max}(R) \times nb_{PS/NU}^{fixe moy min max}(R)$ $=0 \text{ si adressage indirect}$
$nbMax_{lignesTC/page}^{fixe}(*)$	lignes de la TC par page	En mode d'adressage indirect uniquement : nombre maximal (forcément) de lignes de la TC au plus par page : $nbMax_{lignesTC/page}^{fixe}(*) \xrightarrow{\text{ou}} \frac{T_{page}^{fixe}(*)}{T_{ligneTC}^{fixe}(*)}$

Tableau 31. Performances d'accès aux données : nombre « d'informations » par page (d'une relation R)

¹⁰⁶ Le « * » indique que cette donnée ne change pas selon la relation considérée : elle est valable pour toute la BD, voire pour tout le SGBD !

¹⁰⁷ Il faudra « faire des tests » avec l'arrondi par défaut ET avec l'arrondi par excès.

Il est donc maintenant possible de calculer le nombre de pages occupées par une relation R^{108} :

Notation	Unité(s) usuelle(s)	Paramètre (calculé)
$nb_{pagesNU}^{moy min}(R)$	pages	<p>Nombre (moyen ou minimal) de pages de stockage de n-uplets nécessaire pour stocker tous les n-uplets (ET les éventuels pointeurs de suivi en mode d'adressage direct) de la relation R :</p> $nb_{pagesNU}^{moy max}(R) \stackrel{\text{Card}(R)}{=} \frac{Card(R)}{nbMax_{NU/page}^{moy max}(R)}$ <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> Remarque Les éventuels pointeurs de suivi ont déjà été « comptés » : ils ont été pris en compte dans le calcul de nombre de n-uplets par page, il ne faut donc PAS les compter ici une 2^{nde} fois ! </div>
$nb_{pagesTC}^{exact}(R)$	pages	<p>En mode d'adressage indirect uniquement : nombre (forcément exact) de pages nécessaires pour stocker toutes les lignes de la table de correspondance associée à la relation R :</p> $nb_{pagesTC}^{exact}(R) \stackrel{\text{Card}(R)}{=} \frac{nb_{lignesTC}^{exact}(R)}{nbMax_{lignesTC/page}^{fixe}(*)}$
$nb_{pages}^{moy min}(R)$	pages	<p>Nombre (moyen ou minimal) de pages occupées au total pour le stockage des « informations » (i.e. n-uplets et (pointeurs de suivi OU table de correspondance)) de la relation R :</p> $nb_{pages}^{moy min}(R) = nb_{pagesNU}^{moy min}(R) + nb_{pagesTC}^{exact}(R)$ <p style="color: red; text-align: center;"><i>si adressage indirect uniquement</i></p>

Tableau 32. Performances d'accès aux données : nombre de pages occupées (d'une relation R)

On connaît donc le nombre (moyen ou minimal) de pages occupées par une relation R (tout compris)...



Remarque

Il est intéressant de noter que :

- Le calcul des paramètres indépendants d'une relation peut se faire une et une seule fois pour tout le SGBD : ils sont en effet valides pour tout le SGBD !
- De même, le calcul des paramètres liés à une relation R mais indiqués comme « fixes » peuvent être faits une et une seule fois pour la relation R : ils sont en effet valides pour cette relation tant que son modèle conceptuel n'est PAS modifié ET tant que les paramètres généraux de la machine/de l'OS/du SGBD, dont ils peuvent dépendre, ne sont pas modifiés non plus.

¹⁰⁸ Le « * » indique que cette donnée ne change pas selon la relation considérée : elle est valable pour toute la BD, voire pour tout le SGBD !

7 Gestionnaire de mémoire cache

SOMMAIRE DÉTAILLÉ DU CHAPITRE 7

7.1	Principe de la mémoire cache.....	148
7.2	Structure de la mémoire cache.....	149
7.3	Utilisation de la mémoire cache	149
7.3.1	Algorithme de fonctionnement.....	153
7.3.2	Stratégies de rejet de pages de la mémoire cache	155
7.3.3	Stratégies de gestion des conflits de rejets de pages	156
7.3.4	« Punaisage » de pages en mémoire cache	157
7.4	Quid de l'impact sur les performances ?.....	158

FIGURES DU CHAPITRE 7

Figure 102.	Demande d'accès en lecture à une page p SANS mémoire cache	147
Figure 103.	Demande d'accès en écriture à une page p SANS mémoire cache	148
Figure 104.	Structure et contenu de la mémoire cache	149
Figure 105.	Demande d'accès en lecture à une page p AVEC mémoire cache	150
Figure 106.	Demande d'accès en écriture à une page p AVEC mémoire cache	152
Figure 107.	Demande de vidage (total ou, ici, partiel) de la mémoire cache.....	153

TABLEAUX DU CHAPITRE 7

Tableau 33.	Paramètres du gestionnaire de mémoire cache	158
-------------	---	-----

ÉQUATIONS DU CHAPITRE 7

Aucune entrée de table d'illustration n'a été trouvée.

Nous en avons déjà parlé, un SGBD, comme tout logiciel, ne manipule pas directement des données en mémoire de stockage. Ces données doivent être transférées (accès en lecture) depuis la mémoire de stockage vers la mémoire de travail, où elles seront manipulées, avant d'être transférées de nouveau (accès en écriture) en mémoire de stockage si besoin est (*i.e.* si elles ont été modifiées et que cette modification doit être pérennisée). Les demandes d'accès sont faites par les différents « services » du SGBD (cf. §1.3.2) auprès du gestionnaire d'accès aux données (cf. §6) : c'est ce dernier qui réalise effectivement les transferts :

- Les transferts en lecture (*i.e.* les **chargements**) :

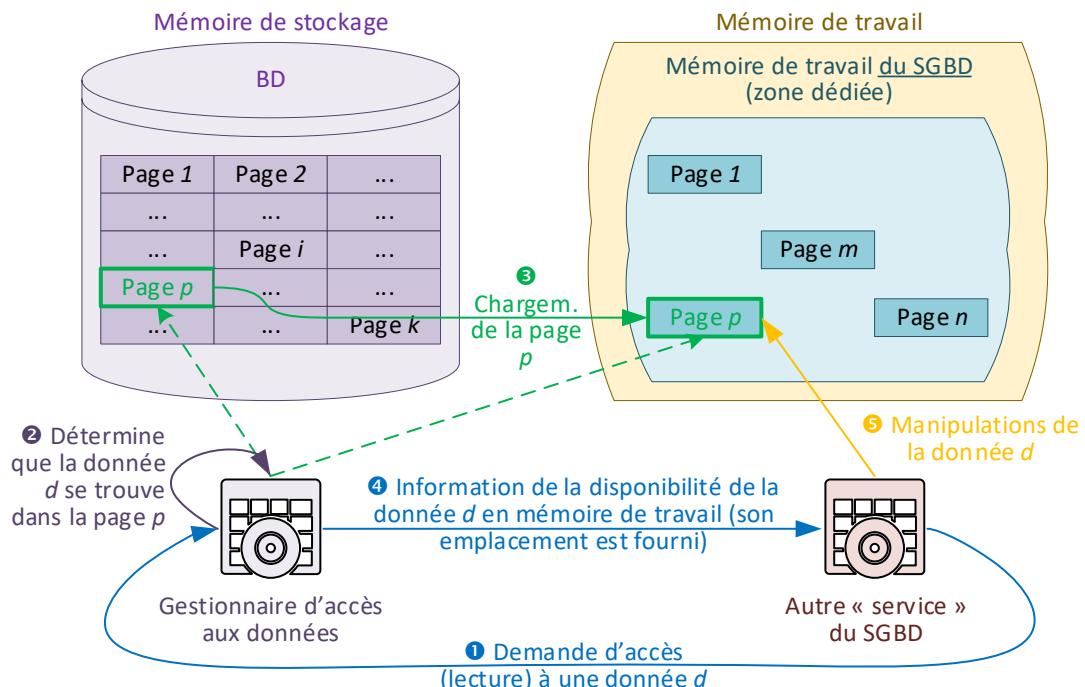


Figure 102. Demande d'accès en lecture à une page p SANS mémoire cache¹⁰⁹

¹⁰⁹ Pour mémoire, nous avons supposé que la taille d'une page est égale à celle d'un bloc. Si un bloc avait contenu plusieurs pages, on aurait dû faire apparaître ces blocs dans cette représentation.

- Les transferts en écriture (*i.e.* les **déchargements**) :

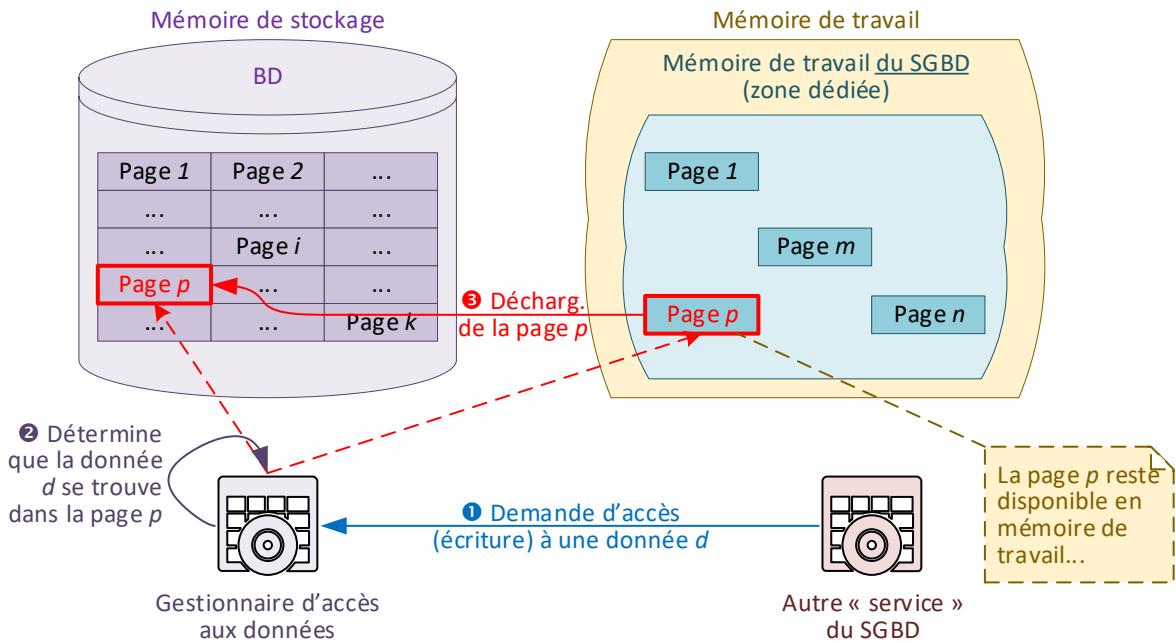


Figure 103. Demande d'accès en écriture à une page p SANS mémoire cache¹⁰⁹

7.1 Principe de la mémoire cache

On l'a déjà dit (cf. §5.1.3), ce sont les transferts mémoire de stockage \leftrightarrow mémoire de travail qui sont coûteux¹¹⁰ : il faut les minimiser. À cette fin, les SGBD mettent généralement en place une **mémoire cache** (**mémoire tampon**, *buffer memory* en anglais¹¹¹) placée en mémoire de travail (au sein de la zone allouée par l'OS au SGBD) pour optimiser (minimiser) le nombre de transferts de pages entre la mémoire de stockage et la mémoire de travail. C'est un mécanisme particulier du SGBD, le **gestionnaire de mémoire cache**, qui est chargé de gérer la mémoire cache dans cet objectif. Les demandes d'accès aux données (lectures et écritures) sont alors prises en charge de concert par le gestionnaire d'accès aux données et le gestionnaire de mémoire cache et se font au travers de cette zone particulière de la mémoire de travail qu'est la mémoire cache. Les principaux objectifs du gestionnaire de mémoire cache sont :

- De rendre les pages et les n-uplets qu'elles contiennent accessibles en mémoire de travail¹¹²,
- De coordonner les accès aux pages en mémoire de stockage en coopération avec le gestionnaire d'accès physique aux données (cf. §6), le gestionnaire de transactions (cf. §3) et le gestionnaire de reprise (cf. §4),
- De minimiser le nombre d'accès à la mémoire de stockage (donc de transferts de pages mémoire de stockage \leftrightarrow mémoire de travail) en essayant d'en « remplacer » le plus possible par des accès à la mémoire de travail (*via* des transferts de pages mémoire de travail \leftrightarrow mémoire de travail), à coût considéré négligeable,
- Tout ceci en travaillant de concert avec le gestionnaire d'accès aux données (cf. §6).

¹¹⁰ Nous avons indiqué (cf. §5.1.3) que nous considérions négligeables les coûts des transferts mémoire de travail \leftrightarrow mémoire de travail et que nous n'aurions pas à nous occuper de transferts mémoire de stockage \leftrightarrow mémoire de stockage.

¹¹¹ Ou, plus simplement « cache » (« tampon » ou « buffer »).

¹¹² Et c'est heureux ! 😊

7.2 Structure de la mémoire cache

Une mémoire cache est donc une partie de la zone de la mémoire de travail dédiée au SGBD. Ce dernier réserve en effet une partie de la mémoire de travail qui lui est allouée pour en faire une mémoire cache. La mémoire cache est une suite contigüe¹¹³ de cases dont chacune peut contenir exactement une page. On peut la voir comme un tableau pouvant stocker n pages et indicé de 1 à n . À tout instant, le gestionnaire de mémoire cache sait quelle page i se trouve dans la case j de la mémoire cache (s'il y en a une : si cette case j est vide, il le sait également !). De façon inverse, il sait si une page p se trouve dans une des cases de la mémoire cache (et, si c'est le cas, il sait dans quelle case elle se trouve).



Définition : « mémoire cache » ou « mémoire tampon » (« cache » ou « tampon »)

La **mémoire cache** une zone de la mémoire de travail allouée au SGBD. Elle est structurée en cases contigües pouvant contenir chacune exactement 1 page. *Via* le gestionnaire de mémoire cache, on peut faire un lien entre une case c de la mémoire cache et une page p (cf. Figure 104) si la case c contient effectivement la page p .

Indice	1	...	c	...	n
Contenu	-	Page i	Page j	Page k	-

Figure 104. Structure et contenu de la mémoire cache



Remarque

On peut stocker en mémoire cache tous types de pages : des pages de stockage de n-uplets mais aussi des pages de stockage de tables de correspondance, des pages de stockage d'index, ...

7.3 Utilisation de la mémoire cache

La mémoire cache peut donc être vue comme un « disque virtuel » (une mémoire de stockage virtuelle) stocké en mémoire centrale (donc d'accès bien plus rapide que les mémoires de stockage « réelles »). Ainsi, les demandes de lecture et d'écriture de pages depuis/sur le disque faites au gestionnaire d'accès aux données sont réalisées en coordination avec le gestionnaire de mémoire cache. Les pages sont alors lues ou écrites depuis/sur la mémoire cache, charge au gestionnaire de mémoire cache de placer dans cette zone mémoire les pages nécessaires au moment où elles sont demandées et d'écrire en mémoire de stockage les pages qui auront été modifiées dans la mémoire cache au moment opportun.

¹¹³ C'est en fait une hypothèse simplificatrice que nous prenons ici : il n'y a en réalité aucune obligation à cette « continuité » de la mémoire cache en mémoire de travail.

Ainsi, l'opération d'accès en lecture à une donnée d se déroule grossièrement comme suit (cf. Figure 105) :

- Une demande d'accès en lecture à une donnée d est transmise au gestionnaire d'accès aux données par un « service » quelconque du SGBD (étape ①) : celui-ci détermine dans quelle page p la donnée d se trouve (étape ②),
- Le gestionnaire d'accès aux données demande au gestionnaire de mémoire cache une demande de placement de la page p en mémoire cache (étape ③),
- Le gestionnaire de mémoire cache fait « ce qu'il faut » pour que la page p existe en mémoire cache et soit copiée en mémoire de travail (étapes ④, puis ⑤a ou [⑤b, ⑤b' et ⑤b'']),
- Le gestionnaire de mémoire cache retourne au gestionnaire d'accès aux données la case d'indice c de la mémoire cache dans laquelle la page p est placée et la zone de la mémoire de travail où elle a été copiée (étape ⑥),
- Le gestionnaire d'accès aux données en informe le « service » du SGBD demandeur de la donnée d (étape ⑦) qui peut alors manipuler cette donnée d en mémoire de travail (étape ⑧).

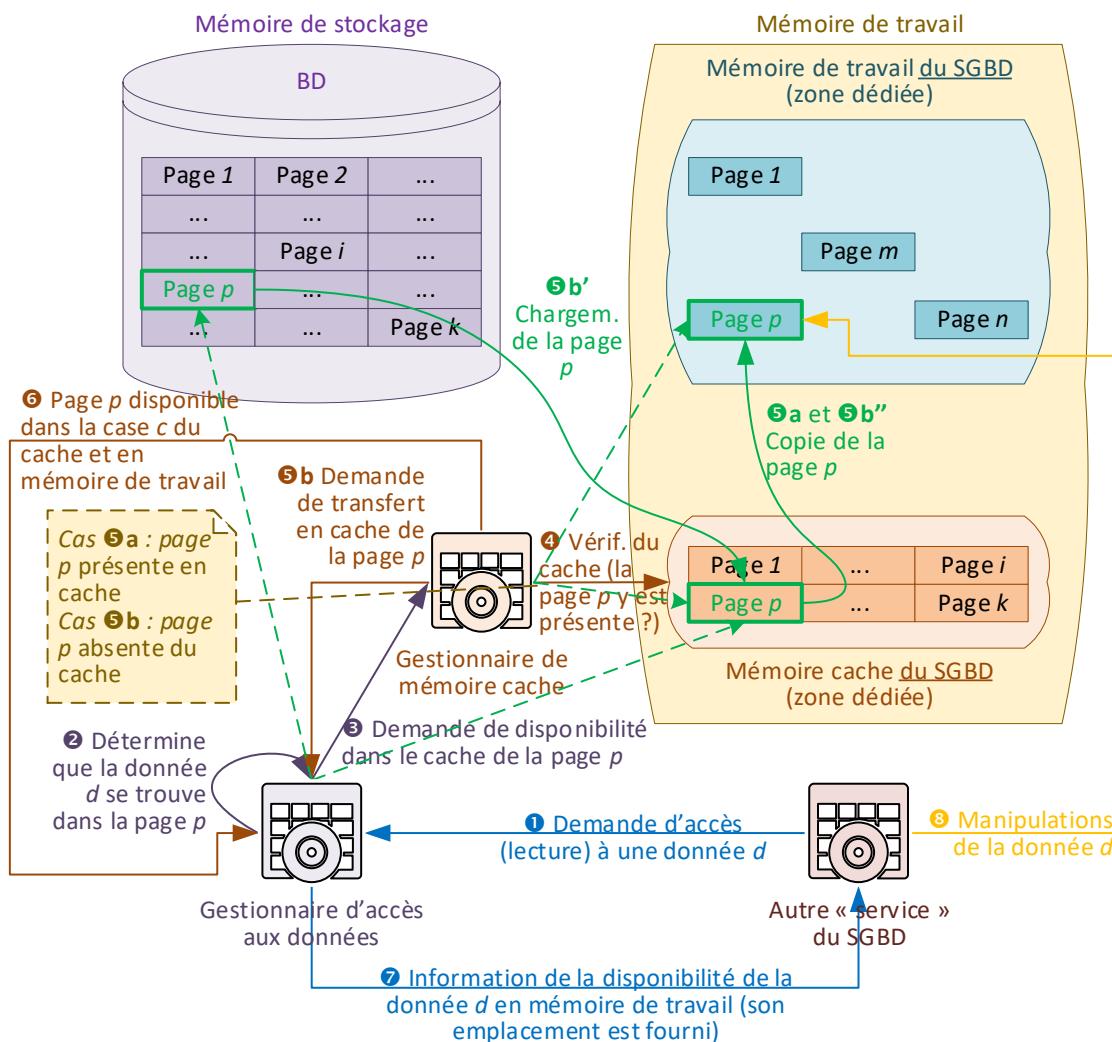


Figure 105. Demande d'accès en lecture à une page p AVEC mémoire cache



Remarque

Notez dans la figure précédente (cf. Figure 105) que 2 cas de figure se présentent à l'étape ❸¹¹⁴ : l'idée de l'utilisation de la mémoire cache est d'essayer de se ramener le plus souvent¹¹⁵ au cas ❸a, où la page p cherchée est déjà présente en mémoire cache. En effet, dans ce cas, on ne fait AUCUN chargement, donc aucun transfert de page mémoire de stockage → mémoire de travail !



Question

Cela est beaucoup plus long (il y a plus d'étapes) qu'un chargement de page sans mémoire cache (cf. Figure 102). Quel est l'intérêt ?

Réponse

Cela fait effectivement plus d'étapes, c'est vrai. Mais cela ne veut pas dire que c'est forcément plus long ! On va même plutôt obtenir globalement de meilleures performances (même si, ponctuellement, *i.e.* pour certaines pages, le chargement pourra effectivement être plus long) : ça tombe bien, c'est l'effet recherché...

De même, l'opération d'accès en écriture à une donnée d se déroule grossièrement comme suit (cf. Figure 106) :

- Une demande d'accès en écriture d'une donnée d est transmise au gestionnaire d'accès aux données par un « service » quelconque du SGBD (étape ❶) : celui-ci détermine dans quelle page p la donnée d se trouve (étape ❷),
- Le gestionnaire d'accès aux données demande au gestionnaire de mémoire cache de réaliser une écriture de la page p (étape ❸),
- Le gestionnaire de mémoire cache copie la page p depuis la mémoire de travail dans la mémoire cache (étape ❹),
- Le gestionnaire de mémoire informe le gestionnaire d'accès aux données que « l'écriture » est réalisée (étape ❺) et ce dernier informe le « service » demandeur que c'est fait (étape ❻).

¹¹⁴ En fait 4, comme nous allons le montrer, mais 3 d'entre eux sont des dérivés du cas ❸b présenté dans la Figure 105.

¹¹⁵ Forcément, on n'y arrivera pas toujours !

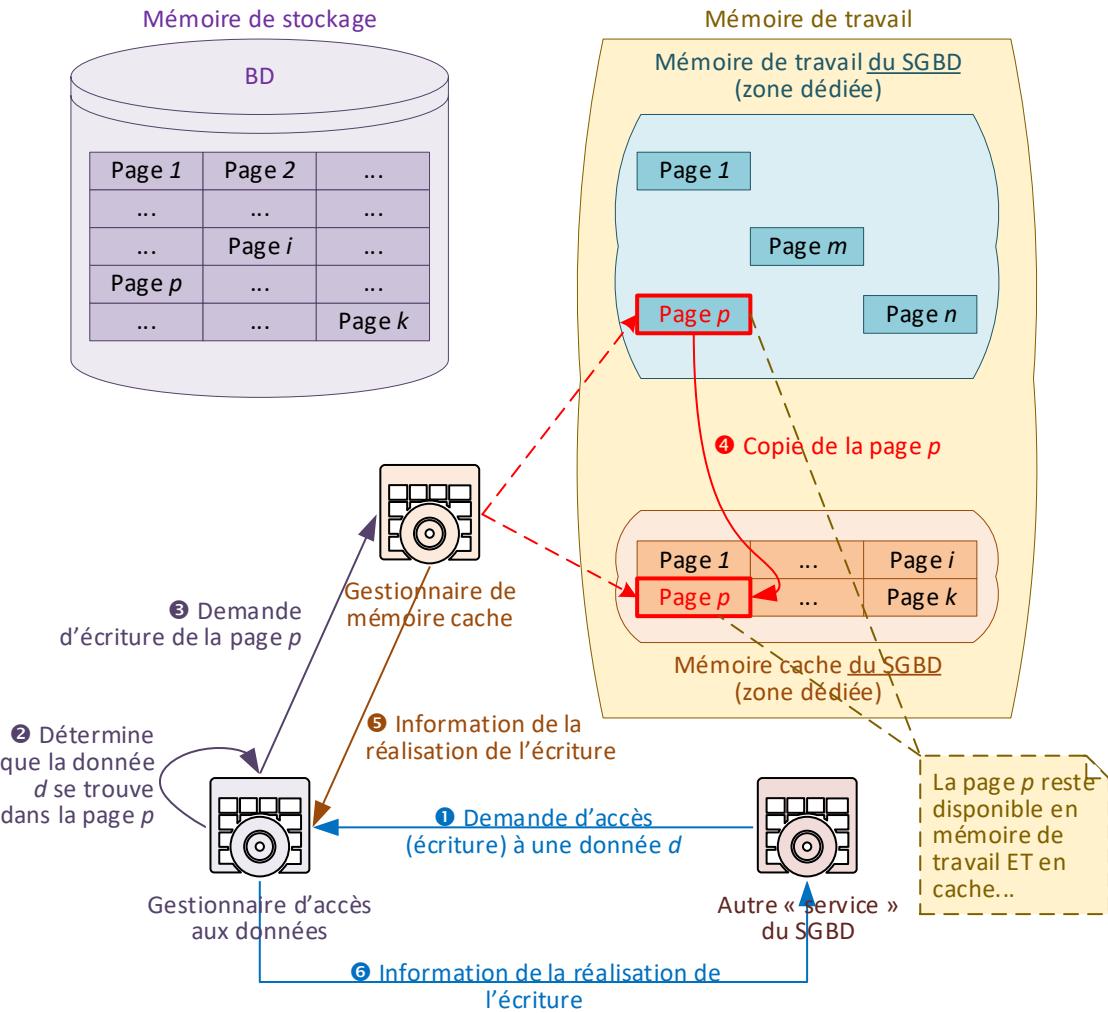


Figure 106. Demande d'accès en écriture à une page p AVEC mémoire cache



Remarque

Notez dans la figure ci-dessus (cf. Figure 106), qu'on ne fait AUCUN déchargement, donc aucun transfert de page mémoire de travail → mémoire de stockage !

Cette dernière remarque amène une question essentielle...



Question

OK, mais alors, les pages modifiées sont sauvegardées à quel moment en mémoire de stockage ?

Réponse

C'est le gestionnaire de mémoire cache (et lui seul¹¹⁶) qui décide des transferts de pages en écriture, lorsqu'il en a besoin (ou à l'extinction du SGBD ou à la fermeture de la BD, bien sûr !). On parle alors de **vidage** (*flushing* en anglais, partiel ou total) du cache. Ce n'est QUE lors de ce vidage que les pages contenues dans le cache (tout ou partie d'entre elles) sont déchargées en mémoire de stockage.

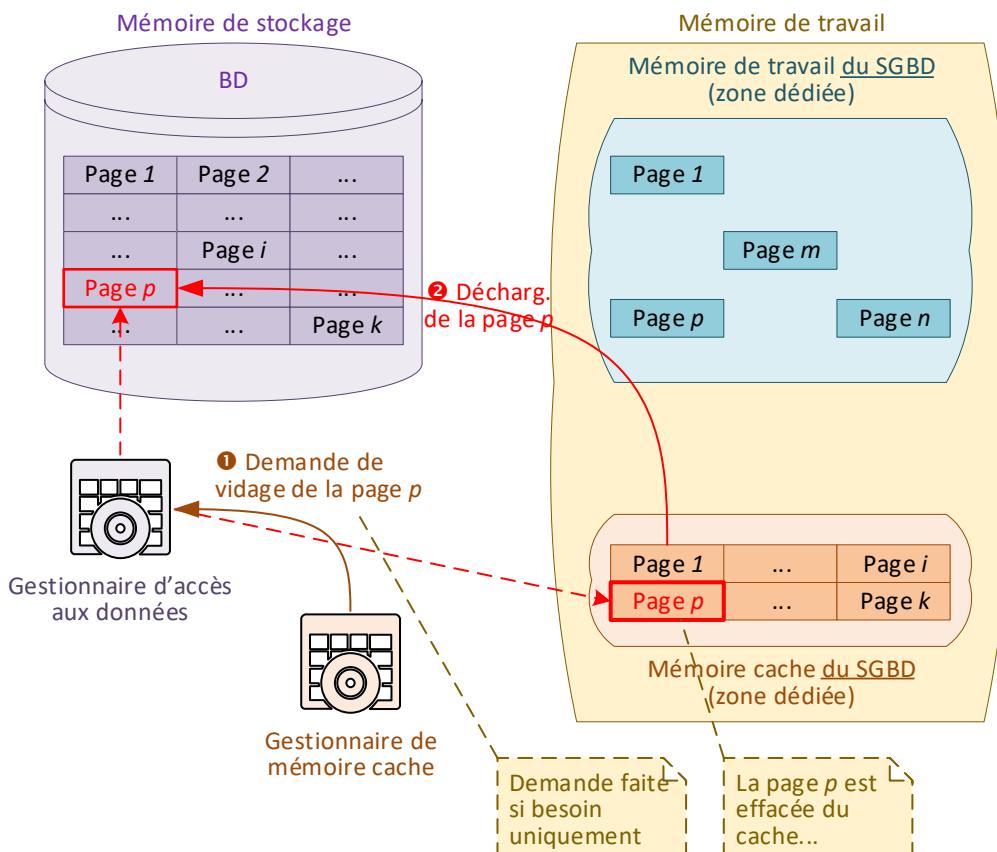


Figure 107. Demande de vidage (total ou, ici, partiel) de la mémoire cache

7.3.1 Algorithme de fonctionnement

Le tout est donc de savoir comment s'assurer de la présence de la page p en mémoire cache. Pour s'assurer de cela, le gestionnaire de mémoire cache va notamment exécuter une consultation du contenu de la mémoire cache et, si besoin, des opérations d'accès (lecture voire écriture) entre la mémoire cache et la mémoire de travail. Dans ce travail, on distingue particulièrement l'opération dite de « rejet de page », **en plus des opérations de chargement et de déchargement**.

¹¹⁶ Sauf opérations très particulières (notamment pour la tenue du journal de reprise après panne, cf. §4.5.1).



Définition : « rejet de page »

L'opération de **rejet de page** est une opération qui consiste à effacer une page du contenu de la mémoire cache (elle est donc effectuée par le gestionnaire de mémoire cache). Si la suppression concerne une page qui a été modifiée depuis son chargement dans la mémoire cache, elle comprend alors également une opération d'écriture (décharge) sur la mémoire de stockage de la page à rejeter.



En pratique : demande d'accès à une page p via la mémoire cache

Le pseudo-algorithme mis en œuvre par le gestionnaire de mémoire cache permettant d'accéder à une page p est le suivant...

Pseudo-algorithme AccèsMémoireCache

Données d'entrée

p : un identificateur de page

Données de sortie

c : case du cache contenant la page p

Données intermédiaires

Aucune

Début

Si la page p demandée est déjà dans le cache **alors**

$c \leftarrow$ case laquelle se trouve la page p

Retourner c (**OTP**)

Sinon

S'il y a une ou des cases libres dans le cache **alors**

$c \leftarrow$ une case libre

Transférer la page p dans cette case c (**1TP_{lecture}**)

Retourner c

Sinon

Choisir dans le cache une page à rejeter

(plusieurs stratégies possibles)

$c \leftarrow$ case contenant la page à rejeter

Si la page à rejeter n'a pas été modifiée depuis son chargement dans le cache **alors**

Effacer la page contenue dans c

Transférer la page p dans la case c (**1TP_{lecture}**)

Retourner c

Sinon

Écrire la page à rejeter sur disque (**1TP_{écriture}**)

Effacer cette page de la case c du cache

Transférer la page p dans la case c (**1TP_{lecture}**)

Retourner c

FinSi

FinSi

FinSi

Fin

La page p contenue dans la case c retournée est transférée en mémoire de travail...

Ainsi, avec ce mécanisme de mémoire cache, les transferts de pages sont minimisés :

- Le chargement d'une page (transfert en lecture) n'est réalisé que si c'est nécessaire (*i.e.* si la page n'est pas déjà présente en mémoire cache),
- Le déchargement d'une page rejetée (transfert en écriture) n'est réalisé que si c'est nécessaire (*i.e.* si la page a été modifiée depuis son chargement en mémoire cache).

Reste à savoir, quand la mémoire cache est pleine, quelle page rejeter avant d'y charger une nouvelle page...

7.3.2 Stratégies de rejet de pages de la mémoire cache

Le pseudo-algorithme ci-dessus peut paraître simplissime (et il l'est !) mais garantit déjà une bonne optimisation des transferts de pages faits par le SGBD. Reste une question en suspens...



Question

Lorsqu'une page p doit être rejetée, comment choisir le plus « intelligemment » celle que l'on va rejeter ?

Réponse

Pas au hasard, c'est certain ! Il existe différentes stratégies de rejet de pages et on peut les regrouper en 2 familles.

Les deux grandes familles de stratégies de rejet de page sont les suivantes :

- La stratégie de rejet LRU (« Least-Recently Used ») : le gestionnaire de mémoire cache rejette la page qui n'a pas été utilisée depuis le plus longtemps,



Remarque

Cette stratégie repose sur l'hypothèse qu'une page inutilisée depuis longtemps n'a que peu de chances d'être réutilisée (au moins dans un avenir proche). On part donc du postulat que les traitements faits sur les données contenues dans une page sont généralement réalisés « dans une période relativement condensée ».

- La stratégie de rejet MRU (« Most-Recently Used ») : au contraire, le gestionnaire de mémoire cache rejette la page qui a été utilisée le plus récemment.



Remarque

Cette stratégie repose sur l'hypothèse qu'une page utilisée récemment n'a que peu de chances d'être réutilisée (au moins dans un avenir proche). On part donc du postulat que les traitements faits sur les données contenues dans une page sont généralement réalisés « d'un bloc ».

Il est important de noter que, implicitement, ces 2 stratégies sont exclusives l'une de l'autre ! De plus, elles nécessitent toutes les 2 de « dater » les utilisations (au moins la dernière utilisation) de chacune des pages contenues dans la mémoire cache¹¹⁷. Avec une telle datation : la stratégie de rejet LRU rejette la page dont la « date de dernière utilisation » est la plus ancienne alors que la stratégie de rejet MRU rejette la page dont la « date de dernière utilisation » est la plus récente.

¹¹⁷ Usuellement, un horodatage (*timestamp*) est associé à chacune des cases de la mémoire cache. **Cet horodatage n'est bien sûr PAS contenu dans les pages du cache** (on ne change pas les données qu'elles contiennent) : ce sont des tables systèmes du SGBD qui conservent ces informations.



Remarque

Une autre façon de « dater » les usages des pages contenues dans le cache est de réaliser une datation « relative » : les cases du cache sont simplement ordonnées dans leur ordre de dernière utilisation (*i.e.* à chaque fois qu'une page contenue dans la mémoire cache est utilisée, elle est placée, par exemple, en tête de la mémoire cache, dans la 1^{ère} case, et les autres pages sont décalées d'un cran à droite). Avec ce système :

- La stratégie de rejet LRU rejette la dernière page (la plus à droite) du cache,
- La stratégie de rejet MRU rejette la première page (la plus à gauche) du cache.



En pratique : performances des stratégies LRU et MRU

Il est impossible d'indiquer quelle stratégie est la meilleure, même globalement : cela dépend en effet des opérations effectuées, de la taille de la mémoire cache et de tout un tas d'autres « paramètres » secondaires. Ainsi, pour une même BD, la stratégie de rejet LRU peut être la meilleure pour une opération OP_1 (et encore, globalement : il existe peut-être des cas limites où MRU est la plus performante) et la stratégie MRU la meilleure pour une opération OP_2 (et encore, globalement).

7.3.3 Stratégies de gestion des conflits de rejets de pages

Il est courant que 2 pages du cache (ou plus) soient utilisées exactement simultanément. Du coup, il est difficile de choisir « la plus anciennement utilisée » (si on a choisi une stratégie LRU) ou « la plus récemment utilisée » (si on a choisi la stratégie MRU)¹¹⁸ : on parle alors de **conflit de rejets**.



Définition : « conflit de rejets »

Il y a un **conflit de rejets** lorsque plusieurs pages du cache ont la même date de dernière utilisation et que cette date est la plus ancienne des dates d'utilisation des pages du cache (si on a choisi la stratégie de rejet LRU) ou la plus récente des dates d'utilisation des pages du cache (si on a choisi la stratégie de rejet MRU).

Quelle que soit la stratégie de rejet choisie (LRU ou MRU), on peut avoir à résoudre des conflits de rejets. Des stratégies de résolution de ces conflits existent (incompatibles entre elles) :

- *La stratégie de résolution « prédictive » des conflits de rejets* : il s'agit d'analyser l'algorithme qui est mis en œuvre et pour la réalisation duquel on effectue les opérations de chargement/déchargement :
 - Si on arrive à déterminer que certaines des pages en conflit de rejets seront encore utilisées dans la suite de l'algorithme, elles sont conservées dans la mesure du possible,
 - Si on arrive à déterminer que certaines des pages en conflit de rejets ne seront plus utilisées dans la suite de l'algorithme, elles seront rejetées en priorité.
- *La stratégie de résolution « hasardeuse » des conflits de rejets* : on rejette une des pages source du conflit de rejets aléatoirement...

¹¹⁸ Et l'on sait tous que l'indéterminisme est l'ennemi de l'informatique ! 😊



Remarque

Ce problème des conflits de rejets ne se pose pas *stricto sensu* si on a choisi de dater « relativement » les cases du cache en fonction de leur « date » d'utilisation en ordonnant simplement ces cases : dans ce cas, la page la plus anciennement utilisée est toujours la dernière du cache et la page la plus récemment utilisée est toujours la première du cache¹¹⁹. Dans ce cas, si 2 pages (ou plus) sont utilisées simultanément, on les « départage » au hasard et on n'a alors jamais de conflit de rejet (en revanche, le tri incessant de la mémoire cache peut finir par être coûteux alors que, normalement, la gestion de la mémoire cache est censée avoir un coût négligeable).



En pratique

La résolution prédictive fonctionne très bien mais est très coûteuse. La résolution hasardeuse, elle, fonctionne moyennement bien mais ne coûte quasiment rien. C'est pourquoi, même dans de nombreux SGBD évolués, c'est souvent la stratégie de résolution hasardeuse des conflits de rejets qui est mise en œuvre.

7.3.4 « Punaisage » de pages en mémoire cache

La mise en oeuvre des différentes stratégies vues précédemment (LRU ou MRU pour le choix de pages à rejeter, prédictive ou hasardeuse pour la résolution des conflits de rejets) permet, normalement, une bonne gestion de la mémoire cache et, donc, une bonne optimisation du nombre de chargements (transferts de pages de la mémoire de stockage vers la mémoire cache) et déchargements (rejets de pages). Cela dit, cette optimisation est globale : il existe quasi-forcément des cas limites où l'optimisation apportée risque de « s'écrouler », voire d'être carrément contreproductive ! Il est cependant parfois possible d'essayer de « reprendre la main » sur ces cas limites. En effet, certains SGBD offrent parfois la possibilité de « punaiser » des pages en mémoire cache.



Définition : « punaisage de page en mémoire cache »

L'opération de **punaisage de page en mémoire cache** permet de fixer volontairement une page en mémoire cache, et ce tant que le punaisage associé à cette page n'est pas levé. Une page ainsi punaisée n'est jamais rejetée. Conséquemment, une page ainsi punaisée ne rentre jamais en conflit de rejets avec d'autres pages (non punaisées, elles, forcément).

Dans les SGBD offrant la possibilité de punaiser (*to pin*) des pages en mémoire cache, cette opération est normalement¹²⁰ compatible avec toutes les stratégies de choix de pages à rejeter (LRU ou MRU) et de résolution des conflits de rejets (prédictive ou hasardeuse).



En pratique

Par exemple, il est courant de punaiser une ou plusieurs pages des index (cf. §8), pour optimiser leur accès.

¹¹⁹ Ou l'inverse, selon l'ordre de classement.

¹²⁰ Tout dépend bien sûr du SGBD mais il n'y a *a priori* pas de raison que cela ne soit pas le cas.

7.4 Quid de l'impact sur les performances ?

Il est impossible de calculer « en moyenne » ni « au mieux » ni « au pire » le bénéfice de la mise en place d'un mécanisme de mémoire cache : cela ne peut se mesurer que pour chaque cas précis (quels traitements ? Mis en œuvre de quelle façon ? Sur quelle donnée) ? Ces mesures reposent sur un (petit) jeu de paramètres :

Notation	Unité(s) usuelle(s)	Paramètre (fourni)
-	-	Stratégie de rejet de pages (LRU ou MRU).
-	-	Stratégie de gestion des conflits de rejets de pages (prédictive ou hasardeuse).
-	-	Possibilité de punaisage de pages ? (oui ou non)
$T_{cache}^{fixe}(*)$	pages ou cases du cache	Taille (fixe) de la mémoire cache (nombre de pages/cases qu'elle peut contenir au maximum).
$T_{utileDsCache}^{exacte}(*)$	pages ou cases du cache	Nombre (exact) de pages/cases « libres » dans le cache (<i>i.e.</i> non-occupées par des pages punaisées) : $T_{utileDsCache}^{exacte} = T_{cache}^{fixe}(*) - Nb_{pagesPunaisees}^{exact}(*)$

Tableau 33. Paramètres du gestionnaire de mémoire cache

8 Indexation des données

SOMMAIRE DÉTAILLÉ DU CHAPITRE 8

8.1	Notions de base sur les index	162
8.1.1	Type d'un index	162
8.1.2	Groupement d'un index (avec la relation indexée)	163
8.1.3	Contenu d'un index	165
8.1.4	Vie d'un index	169
8.1.5	Paramètres généraux (quelle que soit sa structure) d'un index $I(R, c)$	169
8.2	Structure d'un index	171
8.2.1	Arbres B+	172
8.2.1.1	Organisation générale	172
8.2.1.2	Organisation détaillée	175
8.2.1.2.1	Noeuds terminaux (hors racine si elle est seule) et entrées d'index	175
8.2.1.2.2	Noeuds non-terminaux (hors racine) et informations de circulation	176
8.2.1.2.3	Racine et enregistrements d'index	179
8.2.1.2.4	Pages d'un arbre B+ et de la relation indexée	180
8.2.1.3	Évolution d'un arbre B+ : opérations de manipulation	181
8.2.1.3.1	Recherche d'une entrée d'index	182
8.2.1.3.2	Insertion d'une entrée d'index	183
8.2.1.3.3	Insertion d'une information de circulation dans l'index	184
8.2.1.3.4	Division d'une page d'un arbre B+	185
8.2.1.3.5	Suppression d'une entrée d'index	187
8.2.1.3.6	Suppression d'une information de circulation dans l'index	188
8.2.1.3.7	Fusion de 2 pages d'un arbre B+	189
8.2.1.3.8	Exemple de manipulation d'un index en arbre B+ d'ordre 3	191
8.2.1.4	Performances d'un arbre B+	196
8.2.1.4.1	Minimiser le coût d'usage d'un index I en arbre B+ : bien fixer $e_{max}(I(R, c))$	197
8.2.1.4.2	Considérations générales sur les performances d'un arbre B+	197
8.2.1.4.3	Coût d'une lecture dans « le pire des cas »	198
8.2.1.4.4	Coût d'une lecture dans « le meilleur des cas »	201
8.2.1.5	Paramètres spécifiques à un index $I(R, c)$ en arbre B+	204
8.2.2	Index à accès par hachage	205
8.2.2.1	Introduction à la technique du hachage	205
8.2.2.1.1	Concepts basiques pour le hachage	206
8.2.2.1.2	Un hachage « performant », c'est quoi ?	207
8.2.2.1.3	Techniques de hachage basiques	208
8.2.2.1.3.1	Technique de hachage par division	209
8.2.2.1.3.2	Technique de hachage par pliage (sur b bits)	209
8.2.2.2	Index mono-critère à accès par hachage statique (IMCAHS)	212
8.2.2.2.1	Organisation AVEC répertoire (IMCAHSaR)	212
8.2.2.2.1.1	Organisation	212
8.2.2.2.1.2	Évolution d'un IMCAHSaR : opérations de manipulation	215
8.2.2.2.1.2.1	Recherche d'une entrée d'index	216
8.2.2.2.1.2.2	Insertion d'une entrée d'index	217
8.2.2.2.1.2.3	Suppression d'une entrée d'index	218
8.2.2.2.1.2.4	Exemple de manipulation d'un IMCAHSaR	218
8.2.2.2.1.3	Performances d'un IMCAHSaR	225
8.2.2.2.2	Organisation SANS répertoire (IMCAHSsR)	227
8.2.2.2.3	Discussion « AVEC » vs « SANS » répertoire	230
8.2.2.3	Index mono-critère à accès par hachage dynamique (IMCAHD)	232
8.2.2.3.1	Organisation	232
8.2.2.3.2	Évolution d'un IMCAHD : opérations de manipulation	236
8.2.2.3.2.1	Recherche d'une entrée d'index	237
8.2.2.3.2.2	Insertion d'une entrée d'index	238
8.2.2.3.2.3	Division d'une page (en deux)	239
8.2.2.3.2.4	Doublement du répertoire de hachage dynamique	240
8.2.2.3.2.5	Exemple de manipulation d'un IMCAHD	241
8.2.2.3.3	Remarques et performances	246

FIGURES DU CHAPITRE 8

Figure 111. Répartition ordonnée des n-uplets quand R est indexée par un index groupé.....	163
Figure 112. Répartition équiprobable des n-uplets quand R est indexée par un index non-groupé	164
Figure 113. Diagramme de classes UML modélisant le contenu d'un index.....	167
Figure 114. Lien entre pages de l'index et de la relation indexée en mode d'adressage direct	167
Figure 115. Lien entre pages de l'index et de la relation indexée en mode d'adressage indirect	168
Figure 116. Exemple d'arbre B	173
Figure 117. Exemple d'arbre B+	173
Figure 118. Exemple de feuilles d'un arbre B+ dont l'ordre $e_{max}(l(R, c))$ vaut 3	176
Figure 119. Allure générale d'un arbre B+.....	178
Figure 120. Pages de stockage d'un arbre B+ et de la relation indexée (adressage direct)	180
Figure 121. Pages de stockage d'un arbre B+ et de la relation indexée (adressage indirect)	181
Figure 122. Illustration (grossière) de l'objectif de la technique du hachage	206
Figure 123. Cas limite de « mauvais hachage » (tailles des partitions très inégales).....	207
Figure 124. Cas limite de « mauvais hachage » (code hachés inutilisés)	208
Figure 125. Exemple d'index mono-critère à accès par hachage statique avec répertoire	213
Figure 126. Répartition des entrées d'index (v, a) en fonction de leur code haché $h(v)$	213
Figure 127. Pages de stockage d'un IMCAHSaR et de la relation indexée (adressage direct).....	214
Figure 128. Pages de stockage d'un IMCAHSaR et de la relation indexée (adressage indirect).....	215
Figure 129. Exemple d'index mono-critère à accès par hachage statique avec répertoire	227
Figure 130. Exemple d'index mono-critère à accès par hachage statique sans répertoire.....	228
Figure 131. Pages de stockage d'un IMCAHSsR et de la relation indexée (adressage direct)	229
Figure 132. Pages de stockage d'un IMCAHSsR et de la relation indexée (adressage indirect)	230
Figure 133. Organisation générale d'un index mono-critère à accès par hachage dynamique	233
Figure 134. Pages de stockage d'un IMCAHD et de la relation indexée (adressage direct)	235
Figure 135. Pages de stockage d'un IMCAHD et de la relation indexée (adressage indirect)	236
Figure 136. Mauvais et bon résultat du doublement du répertoire de hachage dynamique	240

TABLEAUX DU CHAPITRE 8

Tableau 34. Performances d'indexation : paramètres des n-uplets de la relation indexée	169
Tableau 35. Performances d'indexation : paramètres généraux (indépendants de sa structure) de l'index.....	170
Tableau 36. Structures courantes d'index	171
Tableau 37. Relation Dictionnaire illustrant les structures d'index présentées	172
Tableau 38. Raisonnement sur le contenu d'un arbre B+ (pire des cas).....	199
Tableau 39. Raisonnement sur le nombre de noeuds occupés par un arbre B+ (pire des cas)	200
Tableau 40. Raisonnement sur le contenu d'un arbre B+ (meilleur des cas).....	202
Tableau 41. Raisonnement sur le nombre de noeuds occupés par un arbre B+ (meilleur des cas)	203
Tableau 42. Performances d'indexation : paramètres spécifiques au contenu des arbres B+.....	204
Tableau 43. Performances d'indexation : paramètres spécifiques aux performances des arbres B+	205
Tableau 44. Codes hachés utilisés pour l'exemple d'IMCAHSaR	219
Tableau 45. Performances d'indexation : paramètres spécifiques aux IMCAHSaR.....	226
Tableau 46. Bilan des organisations d'IMCAHS	231
Tableau 47. Performances d'indexation : paramètres spécifiques aux IMCAHSaR.....	231
Tableau 48. Bits de poids fort des codes hachés utilisés pour l'exemple d'IMCAHD.....	241
Tableau 49. Performances d'indexation : paramètres spécifiques aux IMCAHD	247

ÉQUATIONS DU CHAPITRE 8

Équation 25. Coût d'une recherche « brute » des n-uplets tel que $c = v$	161
Équation 26. Coût d'une recherche indexée des n-uplets tels que $c = v$	162
Équation 27. Coût du chargement des n-uplets $c = v$ quand l'index $l(R, c)$ est groupé	163
Équation 28. Coût du chargement des n-uplets $c = v$ quand l'index $l(R, c)$ est non-groupé.....	164
Équation 29. Nombre d'entrées d'index dans un index primaire	166
Équation 30. Taille de la liste d'adresses logiques des entrées d'un index primaire	166
Équation 31. Nombre d'entrées d'index dans un index secondaire	166
Équation 32. Taille de la liste d'adresses logiques des entrées d'un index secondaire	166
Équation 33. Remplissage des feuilles (hors racine) d'un arbre B+	176
Équation 34. Remplissage des noeuds non-terminaux (hors racine) d'un arbre B+	177
Équation 35. Remplissage de la racine d'un arbre B+ (quand elle est son seul noeud).....	179
Équation 36. Remplissage de la racine d'un arbre B+ (quand elle n'est pas son seul noeud)	179
Équation 37. Nombre total d'enregistrements d'index au niveau n d'un arbre B+ (pire des cas)	198
Équation 38. Nombre d'entrées d'index dans les feuilles d'un arbre B+ (pire des cas).....	199
Équation 39. Niveaux et hauteur d'un arbre B+ pour y stocker N entrées d'index (pire des cas)	199
Équation 40. Nombre total d'enregistrements d'index au niveau n d'un arbre B+ (meilleur des cas)	201
Équation 41. Nombre d'entrées d'index dans les feuilles d'un arbre B+ (meilleur des cas)	202
Équation 42. Niveaux et hauteur d'un arbre B+ pour y stocker N entrées d'index (meilleur des cas).....	202

Lorsqu'on manipule une BD, on a très fréquemment besoin d'accéder aux n-uplets possédant une valeur v donnée¹²¹ pour un constituant c d'une relation.



Définition : « constituant d'une relation »

Un **constituant d'une relation** est un sous-ensemble de ses attributs. Un constituant comprend ainsi de 1 à n attributs d'une relation (qui en contiendrait n au total). Un constituant n'est pas forcément la clé primaire de la relation (mais, à l'inverse, la clé primaire d'une relation est, forcément, un constituant de cette relation).

De façon « brutale », chercher dans une BD tous les n-uplets de la relation R pour lesquels la valeur du constituant c vaut v revient, au niveau de l'accès physique aux données, à charger toutes les pages contenant les n-uplets de la relation R ¹²². Ce n'est qu'à l'issue du chargement de toutes ces pages que l'on est sûr d'avoir tous les n-uplets recherchés (ou, à l'inverse, qu'on est sûr qu'il n'en existe aucun).



En pratique

Le coût de la recherche dans les n-uplets de la relation R de tous les n-uplets tels que la valeur de leur constituant c soit la valeur v est le suivant :

$$\text{Coût}_{\text{rechercheNU}}^{\text{moy|min}}(R, c = v) \leq nb_{\text{pages}}^{\text{moy|min}}(R), \forall v$$

Équation 25. Coût d'une recherche « brute » des n-uplets tel que $c = v$

Cette tâche étant, encore une fois, très fréquente et, finalement, très coûteuse (en nombre de transferts de pages), dans l'optique d'accéder les plus rapidement possible (et, donc, de la façon la moins coûteuse possible) à des données (*i.e.* aux n-uplets) ciblées (*i.e.* dont le constituant c vaut la valeur v), toute BD relationnelle peut être munie d'un (ou plusieurs, un par constituant « concerné ») **index** permettant d'associer chaque valeur d'un constituant c d'une relation à l'ensemble des adresses logiques des n-uplets de la BD possédant cette valeur v pour le constituant c indexé.



Définition : « index »

Un **index** $I(R, c)$ est défini sur un constituant c d'une relation R : il associe à tout instant à chaque valeur v de ce constituant c l'ensemble des adresses logiques de tous les n-uplets de la relation R pour lesquels leur constituant c vaut la valeur v .

Ainsi, avec un index $I(R, c)$, si on veut récupérer tous les n-uplets de la relation R possédant la valeur v pour le constituant indexé c , il suffit alors :

1. *Dans l'index de la relation R associé au constituant c* : de chercher en son sein les adresses logiques des n-uplets associés à la valeur v du constituant indexé c , (**opération la moins coûteuse possible, ce coût, aussi faible soit-il, étant la grande majorité du temps largement compensé par le gain réalisé à l'étape suivante**),
2. *Dans les pages de stockage des n-uplets de la relation R* : de transférer (charger) uniquement les pages contenant des n-uplets dont on a récupéré l'adresse logique à l'étape précédente (**ainsi, aucune des pages ne contenant aucun de ces n-uplets n'est transférée pour rien, d'où une économie de transferts de pages, souvent importante en regard du coût d'usage de l'index à l'étape précédente**).

¹²¹ Ou une valeur v comprise dans un intervalle de valeurs, ce qui n'est finalement guère différent.

¹²² Sans parler du coût de transfert des pages contenant l'éventuelle table de correspondance (si elle ne se trouve pas déjà intégralement en mémoire de travail) associée à R si on est en mode d'adressage indirect.



En pratique

Pour un index $I(R, c)$, le coût de la recherche dans les n-uplets de la relation R de tous les n-uplets tels que la valeur de leur constituant c soit la valeur v est le suivant :

$$\text{Coût}_{\text{rechercheNU}}(R, c = v) = \frac{\text{Coût}_{\text{usage}}^{\text{fixe|moy|min |max}}(I(R, c))}{\text{selon structure (cf. §8.2)}} + \frac{\text{Coût}_{\text{chargementNU}}^{\text{exact|moy|min}}(R, c = v)}{\text{selon groupement (cf. §8.1.2)}}$$

Équation 26. Coût d'une recherche indexée des n-uplets tels que $c = v$



Attention

Il est inutile, voire même contreproductif, d'indexer tous les constituants d'une relation R : la place occupée par les index et leur « maintenance » va faire perdre des performances, surtout pour ceux qui définis sur des constituants c rarement utilisés.

8.1 Notions de base sur les index

Nous venons de le voir, dans une BD relationnelle, un index $I(R, c)$ est toujours défini par rapport à un constituant c d'une relation R (la relation est alors dite « indexée »). Il existe pas mal de structures différentes d'index. Tous les index peuvent néanmoins tous être caractérisés, notamment, par leur type et leur groupement avec la relation indexée et, donc, par leur structure.

8.1.1 Type d'un index

Si le constituant indexé c est la clé primaire de la relation, on parlera d'**index primaire**, sinon on parlera d'**index secondaire**.



Définition : « index primaire »

Un **index primaire** $I(R, c)$ est un index (*sic*) défini par rapport à un constituant c qui est en fait la clé primaire de la relation R indexée. Ainsi, *via* un index primaire, chaque n-uplet de la relation indexée est relié¹²³ à une et une seule valeur v du constituant indexé c et, inversement, chaque valeur v du constituant indexé c est reliée¹²³ à un et un seul n-uplet de la relation indexée R . Il y a donc exactement autant de n-uplets dans la relation indexée R qu'il existe de valeurs différentes du constituant indexé c .



Définition : « index secondaire »

Un **index secondaire** $I(R, c)$ est un index (*sic*) défini par rapport à un constituant c qui n'est PAS la clé primaire de la relation R indexée. Ainsi, *via* un index secondaire, chaque n-uplet de la relation indexée est relié¹²³ à une et une seule valeur v du constituant indexé c mais, en revanche, chaque valeur v du constituant indexé c est reliée¹²³ à un ou plusieurs n-uplets de la relation indexée R ¹²⁴. Il y a donc un nombre de n-uplets dans la relation indexée R supérieur ou égal au nombre de valeurs différentes du constituant indexé c pour cette relation R .

¹²³ Le lien est en réalité fait *via* l'adresse logique des n-uplets.

¹²⁴ Dans un index secondaire, la liste des adresses logiques associée à une entrée d'index est appelée **liste inverse**.

Par définition, une relation R n'ayant qu'une seule clé primaire, il est possible de définir 1 index primaire au maximum sur cette relation R^{125} . Au contraire, une relation R pouvant avoir de 0 à n constituants non-clé, il est possible de définir de 0 à n index secondaires sur cette relation R^{126} .

8.1.2 Groupement d'un index (avec la relation indexée)

On y reviendra, un index est stocké dans des pages (comme tout dans une BD en fait !). À ce niveau, on distingue les **index groupés** (sous-entendu « avec la relation qu'ils indexent ») et les **index non-groupés** (avec le même sous-entendu). Le fait de grouper (ou non) un index avec la relation qu'il indexe est important : cela a notamment un impact (ou non) sur le « rangement » des n-uplets de cette relation dans les pages de stockage de ces n-uplets.

Définition : « index groupé »

Un **index groupé** $I(R, c)$ est un index dont l'existence-même impacte le « rangement » des n-uplets de la relation indexée R au sein de leurs pages de stockage. Pour un index groupé $I(R, c)$, alors, dans les pages de stockage des n-uplets de la relation R , tous ces n-uplets sont ordonnées par « groupes » de mêmes valeurs v du constituant indexé c .

Page p		Page p'		Page p''	
NU($c = v_1$)	NU($c = v_1$)	NU($c = v_3$)	NU($c = v_3$)	NU($c = v_7$)	NU($c = v_7$)
NU($c = v_1$)	NU($c = v_1$)		NU($c = v_4$)	NU($c = v_7$)	NU($c = v_7$)
NU($c = v_1$)	NU($c = v_2$)		NU($c = v_5$)	NU($c = v_7$)	
NU($c = v_2$)	NU($c = v_2$)		NU($c = v_5$)		NU($c = v_8$)
NU($c = v_2$)	NU($c = v_2$)	NU($c = v_6$)		NU($c = v_8$)	NU($c = v_8$)
	NU($c = v_3$)	NU($c = v_7$)	NU($c = v_7$)		NU($c = v_8$)
...

Figure 108. Répartition ordonnée des n-uplets quand R est indexée par un index groupé

Ainsi, tous les n-uplets possédant la même valeur v pour le constituant indexé c^{127} d'un index groupé sont toujours situés les uns à la suite des autres dans les pages de stockage des n-uplets de la relation R . Donc, pour un index groupé, en supposant que les valeurs différentes du constituant indexé c sont réparties de façon équiprobables sur les n-uplets de la relation¹²⁸, on a alors pour le coût (en nombre de transferts de page) du chargement de tous les n-uplets tels que $c = v^{129}$:

$$\text{Coût}_{\text{chargement}_NU}^{\text{moy/min}}(R, c = v) \stackrel{?}{=} \frac{\text{nb}_{\text{pages}}^{\text{moy/min}}(R)}{\text{nb}_{\text{valDiff}}^{\text{exact}}(R, c)}$$

Équation 27. Coût du chargement des n-uplets $c = v$ quand l'index $I(R, c)$ est groupé

¹²⁵ C'est même parfois, selon les SGBD, fait automatiquement lors de la définition de la relation R et de sa clé primaire : dans ces SGBD, une relation est usuellement implicitement indexée.

¹²⁶ Mais il faut savoir rester raisonnable, comme l'indique le cadre « attention » sur la page précédente.

¹²⁷ Forcément, cela n'a de sens que si l'index est secondaire : s'il est primaire, il n'y a qu'un et un seul n-uplet possédant chaque valeur v du constituant indexé c .

¹²⁸ Et que ces pages de stockage sont remplies de façon optimale (*i.e.* le plus possible).

¹²⁹ Notez qu'on ne donne bien ici que le coût de chargement de ces n-uplets, cela ne comprend donc pas le coût de la recherche de leur adresse logique en utilisant l'index groupé.

On voit bien ici l'intérêt de grouper un index $I(R, c)$ avec la relation R qu'il indexe : accélérer (grandement !) la recherche des n-uplets possédant une valeur v donnée pour le constituant indexé c . Cela dit, le groupement d'un index n'est pas « une recette miracle » (ça serait trop beau pour être vrai 😊). Il existe en effet des limites dues à la notion-même d'index :

- *Il est inutile de grouper un index primaire $I(R, c)$ avec la relation R qu'il indexe* : par définition, il existe alors pour chaque valeur du constituant indexé un et un seul n-uplet associé ; il ne sert donc à rien d'ordonner ces n-uplets en fonction de la valeur du constituant indexé afin, notamment, de les « regrouper » par valeurs égales !
- *Il est impossible de grouper plusieurs index secondaires avec une même relation R* : grouper un index $I(R, c)$ impliquerait de regrouper les n-uplets de la relation indexée R en fonction des valeurs du constituant indexé c ; donc, grouper plusieurs index avec la relation R demanderait d'ordonner les n-uplets de cette relation simultanément en fonction des valeurs de tous les constituants indexés, ce qui est, bien sûr, un non-sens¹³⁰ !

On groupe donc, au maximum, un index secondaire $I(R, c)$ avec une relation indexée R . L'éventuel index primaire de cette relation est forcément non-groupé et tous les autres index secondaires de cette relation sont eux aussi non-groupés. Le choix de l'index secondaire à grouper est donc déterminant : on groupera de préférence celui qui indexe le constituant non-clé le plus souvent utilisé.

Définition : « index non-groupé »

Un **index non-groupé** $I(R, c)$ est un index dont l'existence n'impacte pas le « rangement » des n-uplets de la relation indexée R (au sein de leurs pages de stockage).



Page p		Page p'		Page p''	
NU($c = v_2$)	NU($c = v_8$)	NU($c = v_1$)		NU($c = v_2$)	NU($c = v_7$)
NU($c = v_8$)	NU($c = v_6$)	NU($c = v_1$)	NU($c = v_7$)		NU($c = v_7$)
NU($c = v_3$)	NU($c = v_2$)		NU($c = v_8$)	NU($c = v_3$)	
NU($c = v_7$)	NU($c = v_1$)	NU($c = v_5$)	NU($c = v_2$)	NU($c = v_4$)	NU($c = v_5$)
NU($c = v_5$)	NU($c = v_7$)	NU($c = v_7$)	NU($c = v_3$)		NU($c = v_8$)
NU($c = v_1$)			NU($c = v_7$)	NU($c = v_2$)	NU($c = v_1$)

Figure 109. Répartition équiprobable des n-uplets quand R est indexée par un index non-groupé

Ainsi, tous les n-uplets possédant la même valeur v pour le constituant indexé c d'un index non-groupé $I(R, c)$ sont répartis « aléatoirement un peu partout » dans les pages de stockage des n-uplets de la relation R . On considérera alors que leur répartition est équiprobable. On a alors pour le coût (en nombre de transferts de pages en lecture) du chargement de tous les n-uplets de R tels que $c = v$ ¹³¹ :

$$\text{Coût}_{\text{chargementNU}}^{\text{exact}}(R, c = v) \stackrel{?}{=} \frac{\text{Card}(R)}{nb_{valDiff}^{\text{exact}}(R, c)}$$

Équation 28. Coût du chargement des n-uplets $c = v$ quand l'index $I(R, c)$ est non-groupé

¹³⁰ Il est en effet impossible de grouper un même ensemble simultanément selon 2 valeurs différentes.

¹³¹ Notez que, là encore, on ne donne bien ici que le coût de chargement de ces n-uplets, cela ne comprend donc pas le coût de la recherche de leur adresse en utilisant l'index non-groupé.

8.1.3 Contenu d'un index

Comme tout, un index est stocké en mémoire de stockage dans des pages. Ces dernières ont une organisation et un contenu qui leur sont propres¹³². Le contenu d'un index (et, donc, de « ses » pages) varie en fonction de sa structure mais, invariablement, ce qu'on trouve majoritairement dans un index ce sont des **entrées d'index**. On y trouve aussi des **informations de circulation** dans l'index.

Ce sont les entrées d'index qui forment le lien entre les différentes valeurs du constituant indexé c et les n-uplets possédant la valeur correspondante pour ce constituant.



Définition : « entrée d'index »

Une **entrée d'index** fait le lien entre une valeur v du constituant indexé et les adresses logiques de tous les n-uplets possédant cette valeur-là pour ce constituant. Une entrée d'index est donc toujours un couple associant une **valeur v** du constituant indexé à une **liste d'adresses logiques a** (cette liste contient les adresses logiques de tous les n-uplets pour lesquels le constituant indexé c vaut la valeur v).

L'intérêt est de pouvoir accélérer les opérations d'accès à des n-uplets :

- Si on cherche tous les n-uplets possédant une certaine valeur v pour le constituant c sur lequel un index est construit, il suffit de trouver l'entrée d'index dont la valeur v_{EI} est égale à celle cherchée pour connaître les adresses logiques de tous les n-uplets recherchés,
- Si on cherche tous les n-uplets dont la valeur v pour un constituant c sur lequel un index est construit appartient à une plage de valeurs, il suffit de trouver toutes les entrées d'index dont la valeur v_{EI} appartient à la plage de valeurs donnée pour connaître les adresses logiques de tous les n-uplets recherchés.



Rappel

La forme des adresses logiques contenues dans la liste d'adresses a de chaque entrée d'index dépend bien sûr du mode d'adressage de la BD :

- *En mode d'adressage direct* : chaque adresse logique a la forme d'un couple (p, i) où p est un identificateur de page et i est l'indice dans une case du répertoire des déplacements situé à la fin de cette page,
- *En mode d'adresse indirect* : chaque adresse logique a la forme d'un identifiant logique id .

En prenant en compte tous ces points, on remarque bien qu'un index $I(R, c)$ « vit » en fonction de l'évolution de la relation R qu'il indexe, notamment :

- Tout ajout d'un n-uplet dans la relation R provoque l'ajout d'une adresse logique (celle du n-uplet ajouté) à une entrée d'index dans $I(R, c)$, voire à l'ajout d'une entrée d'index dans $I(R, c)$,
- Toute suppression d'un n-uplet dans la relation R provoque la suppression d'une adresse logique (celle du n-uplet supprimé) à une entrée d'index dans $I(R, c)$, voire à la suppression d'une entrée d'index dans $I(R, c)$,
- Toute modification d'une ou de plusieurs des valeurs d'attributs (faisant partie du constituant indexé) d'un n-uplet dans la relation R provoque des modifications/ajouts/suppressions dans les entrées d'index de $I(R, c)$.

¹³² L'organisation interne et le contenu de ces pages n'ont donc rien à voir avec l'organisation interne et le contenu des pages de stockage des n-uplets.

Ainsi, selon le type de l'index, on peut implicitement observer plusieurs aspects...

En pratique

Ainsi, si l'index $I(R, c)$ est primaire :

- Il y a autant d'entrées d'index dans l'index $I(R, c)$ que de n-uplets dans la relation R indexée :

$$nb_{entrees}^{exact}(I(R, c)) = Card(R)$$

Équation 29. Nombre d'entrées d'index dans un index primaire

- La taille, en nombre d'adresses logiques, de la liste d'adresses a associée à chaque entrée d'index dans l'index I est de 1 :

$$nb_{adrLogNU/entree}^{fixe}(I(R, c)) = 1$$

Équation 30. Taille de la liste d'adresses logiques des entrées d'un index primaire



Cependant, si l'index $I(R, c)$ est secondaire :

- Le nombre d'entrées d'index est égal au nombre de valeurs différentes du constituant indexé c :

$$nb_{entrees}^{exact}(I(R, c)) = nb_{valDiff}^{exact}(R, c)$$

Équation 31. Nombre d'entrées d'index dans un index secondaire

- La taille, en nombre d'adresses logiques, de la liste d'adresses a associée à chaque entrée d'index dans l'index $I(R, c)$ est égal au nombre de n-uplets de R par valeur du constituant indexé c :

$$nb_{adrLogNU/entree}^{moy}(I(R, c)) \stackrel{\text{?}}{=} \frac{Card(R)}{nb_{valDiff}^{exact}(R, c)}$$

Équation 32. Taille de la liste d'adresses logiques des entrées d'un index secondaire

Usuellement, un index ne contient pas que des entrées d'index. Il contient très souvent d'autres données permettant de trouver au plus vite la ou les entrées d'index recherchées : ce sont les **informations de circulation dans l'index**. Ainsi, globalement, grâce à ces informations de circulation dans l'index, l'optimisation se fait à 2 niveaux :

- Au niveau de la circulation dans l'index lui-même* : afin de trouver le plus rapidement possible, au sein de l'index, la ou les entrées d'index qui nous intéressent,
- Au niveau de l'accès aux n-uplets* : afin de charger le plus vite possible, une fois qu'on a leur adresse logique via la ou les entrées d'index, les n-uplets recherchés.



Définition : « information de circulation dans l'index »

Une **information de circulation dans l'index** est, comme son nom l'indique, une information permettant d'atteindre dans un index le plus rapidement possible la ou les entrées d'index qui nous intéressent. La forme d'une information de circulation est fortement dépendante de celle de l'index (contrairement aux entrées d'index qui ont toujours la même forme, quelle que soit la structure d'index dont on parle).

On parlera d'**enregistrement d'index** pour désigner aussi bien les entrées de l'index que les informations de circulation dans l'index.



Définition : « enregistrement d'index »

Un **enregistrement d'index** désigne aussi bien une entrée d'un index qu'une information de circulation dans cet index. C'est donc un terme générique permettant de désigner tout ce qui peut être contenu dans un index.

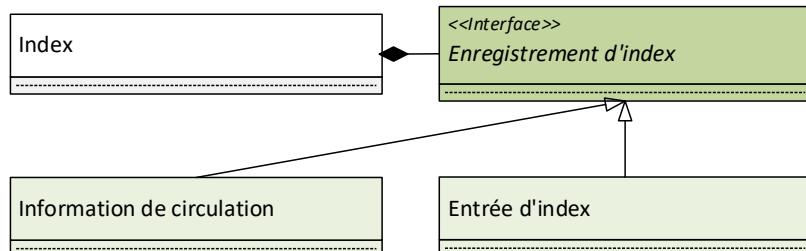


Figure 110. Diagramme de classes UML modélisant le contenu d'un index

Donc, globalement, au niveau de leur stockage dans des pages, on peut voir un index et la relation indexé comme suit :

- Si la relation indexée est en mode d'adressage direct : dans ce cas, le lien (cf. Figure 111) est fait depuis les entrées d'index ; celles-ci associent par définition une valeur v du constituant indexé à la liste a des adresses logiques des n-uplets de la relation indexée possédant cette même valeur v pour le constituant indexé. Puisqu'on est en mode d'adressage direct, chaque adresse logique de cette liste correspond en réalité :
 - Soit à l'adresse physique d'un n-uplet s'il est toujours stocké dans sa page d'origine,
 - Soit à l'adresse physique d'un pointeur de suivi associé à un n-uplet si ce dernier n'est plus stocké dans sa page d'origine.

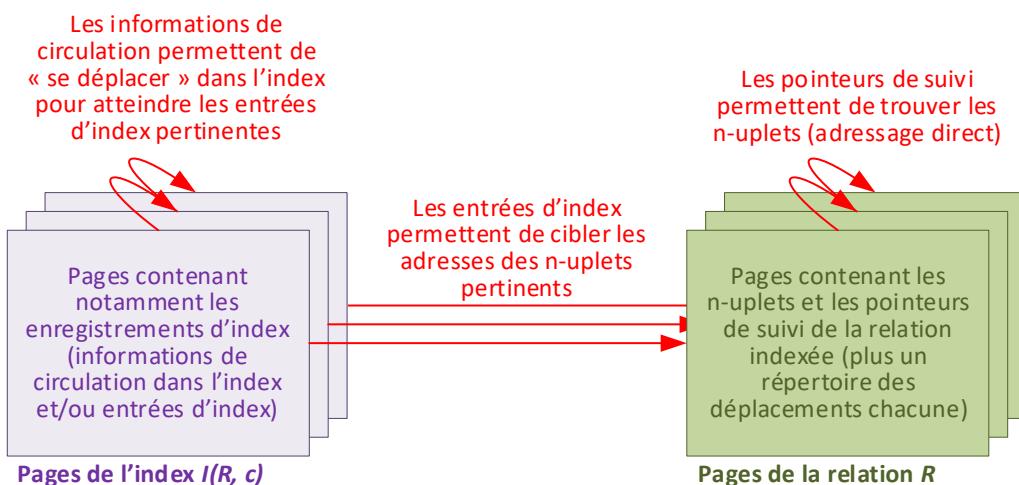


Figure 111. Lien entre pages de l'index et de la relation indexée en mode d'adressage direct

- Si la relation indexée est en mode d'adressage indirect : le lien (cf. Figure 112) est toujours fait depuis les entrées d'index via la valeur v du constituant indexé qu'elles contiennent vers les n-uplets possédant cette même valeur v pour le constituant indexé. Mais, puisqu'on est en mode d'adresse indirect, ces adresses logiques sont en fait des identifiants logiques qui vont, via la table de correspondance de la relation, être associées aux adresses physiques des n-uplets.

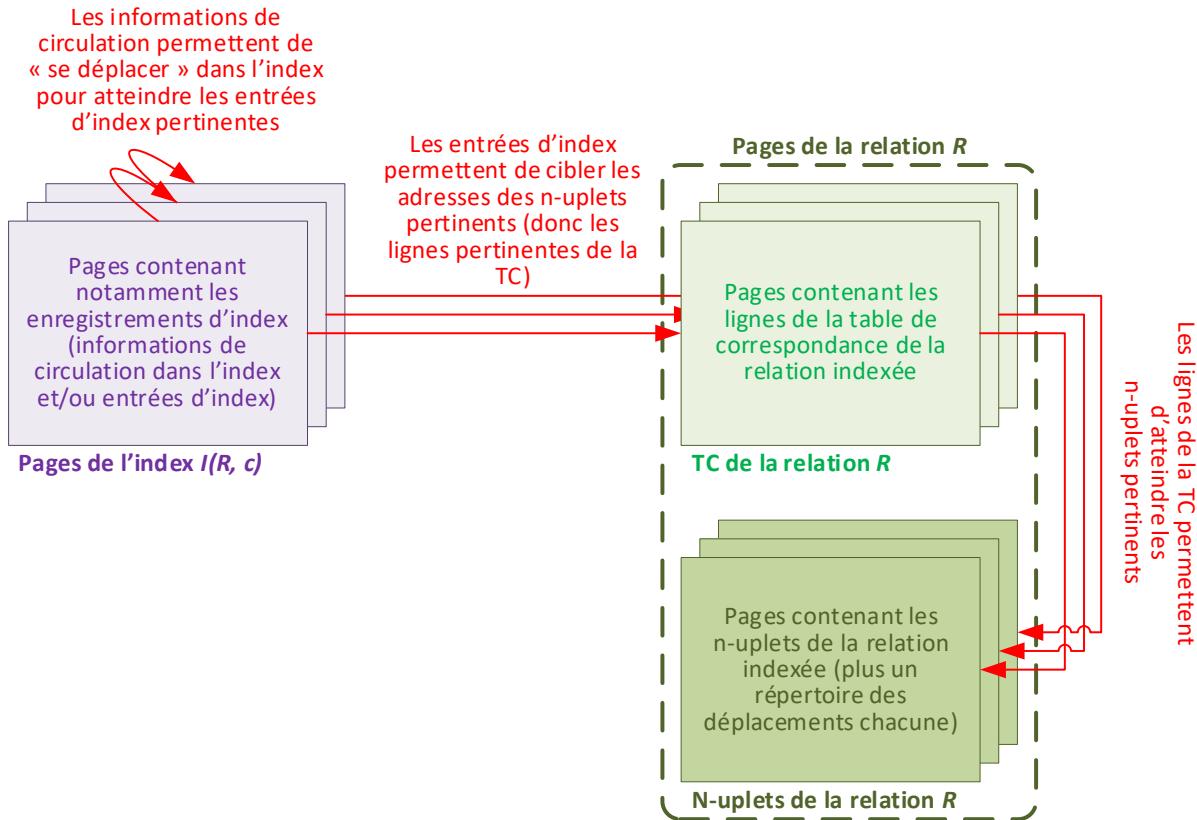


Figure 112. Lien entre pages de l'index et de la relation indexée en mode d'adressage indirect



Remarque

Nous l'avons déjà dit, les pages stockant les n-uplets d'une relation ne sont pas organisées de la même façon que les pages stockant les lignes de l'éventuelle table de correspondance de cette relation. De même, les pages de l'index ne sont, elles non plus, pas organisées de la même façon et leur organisation dépend de la structure-même de l'index (cf. §8.2). **Cependant, sauf exception, les pages contenant les entrées de l'index ne sont pas organisées en interne de la même façon que les pages contenant les informations de circulation dans l'index.**

8.1.4 Vie d'un index

Un index évolue au cours de son existence et cette évolution dépend de celle de la relation indexée :

- *Lorsque la relation indexée est vide* : il n'y a aucun n-uplet à indexer, donc l'index est vide lui aussi et ne contient donc aucune entrée d'index (mais peut contenir des informations de circulation dans l'index, tout dépend de sa structure, cf. §8.2),
- *Lorsque des n-uplets sont ajoutés à la relation indexée* : des entrées d'index sont modifiées et/ou ajoutées (s'il n'existe aucun entrée d'index associée à une des valeurs v du constituant indexé figurant dans les n-uplets ajoutés à la relation) ; parallèlement, des informations de circulation dans l'index peuvent être modifiées et/ou ajoutées (selon la structure de l'index, cf. §8.2),
- *Lorsque des n-uplets sont supprimés de la relation indexée* : les entrées d'index sont modifiées et/ou supprimées (s'il existait une entrée d'index associée à une valeur v du constituant indexé et que le dernier des n-uplets possédant cette valeur v pour ce constituant est supprimé de la relation) ; parallèlement, des informations de circulation dans l'index peuvent être modifiées et/ou supprimées (selon la structure de l'index, cf. §8.2),
- *Lorsque des n-uplets sont modifiés dans la relation indexée* : des entrées d'index sont modifiées et/ou ajoutées et/ou supprimées ; parallèlement, des informations de circulation dans l'index peuvent être modifiées et/ou ajoutées et/ou supprimées (selon la structure de l'index, cf. §8.2).

8.1.5 Paramètres généraux (quelle que soit sa structure) d'un index $I(R, c)$

Les paramètres d'un index $I(R, c)$ suivants sont communs à toutes les structures d'index :

- *Paramètres liés au n-uplets de la relation indexée R* :

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$Cout_{chargementNU}^{exact moy min}(R, c = v)$	pages	<p>Nombre (exact ou moyen ou minimal) de pages à charger pour lire tous les n-uplets d'une relation R tels que leur constituant c vaille une valeur v une fois les adresses logiques de ces n-uplets récupérées dans l'index $I(R, c)$:</p> <ul style="list-style-type: none"> • Si l'index $I(R, c)$ est groupé : $Cout_{chargementNU}^{moy min}(R, c = v) \stackrel{\text{def}}{=} \frac{nb_{pages}^{moy min}(R)}{nb_{valDiff}^{exact}(R, c)}$ <ul style="list-style-type: none"> • Si l'index $I(R, c)$ est non-groupé : $Cout_{chargementNU}^{exact}(R, c = v) \stackrel{\text{def}}{=} \frac{Card(R)}{nb_{valDiff}^{exact}(R, c)}$
$T_{adrLogNU}^{fixe}(*)$	octets	<p>Taille (forcément fixe) d'une adresse logique de n-uplet :</p> <ul style="list-style-type: none"> • Si on est en mode d'adressage direct pour le stockage des n-uplets : $T_{adrLogNU}^{fixe}(*) = T_{idPage}^{fixe}(*) + T_{idCaseDepl}^{fixe}(*)$ <ul style="list-style-type: none"> • Si on est en mode d'adressage indirect pour le stockage des n-uplets : $T_{adLogNU}^{fixe}(*) = T_{idLog}^{fixe}(*)$

Tableau 34. Performances d'indexation : paramètres des n-uplets de la relation indexée

- Paramètres liés aux entrées d'index de l'index $I(R, c)$:

Notation	Unité(s) usuelle(s)	Paramètre (fourni)
$nb_{adrLogNU/entree}^{fixe moy}(I(R, c))$	adresses logiques de n-uplets par entrée d'index	<p>Nombre (fixe ou moyen) d'adresses logiques de n-uplets par entrée d'index :</p> <ul style="list-style-type: none"> Si $I(R, c)$ est primaire : $nb_{adrLogNU/entree}^{fixe}(I(R, c)) = 1$ <ul style="list-style-type: none"> Si $I(R, c)$ est secondaire : $nb_{adrLogNU/entree}^{moy}(I(R, c)) \stackrel{\text{Card}(R)}{=} nb_{valDiff}^{exact}(R, c)$
$T_{entree}^{fixe moy min max}(I(R, c))$	octets	<p>Taille (fixe ou moyenne ou minimale ou maximale) d'une entrée d'index dans l'index $I(R, c)$:</p> $T_{entree}^{fixe moy min max}(I(R, c)) = T_{valAtt}^{fixe moy min max}(R, att)$ $+ \left(nb_{adrLogNU/entree}^{fixe moy}(I, (R, c) \times \underbrace{T_{adrLogNU}^{fixe}(*)}_{\text{dépend du mode d'adressage}} \right)$
$nb_{entrees}^{exact}(I(R, c))$	entrées d'index	<p>Nombre (exact) d'entrées dans l'index $I(R, c)$:</p> <ul style="list-style-type: none"> Si l'index est primaire : $nb_{entrees}^{exact}(I(R, c)) = Card(R)$ <ul style="list-style-type: none"> Si l'index est secondaire : $nb_{entrees}^{exact}(I(R, c)) = nb_{valDiff}^{exact}(R, c)$
$nbMax_{entrees/page}^{fixe moy min max}(I(R, c))$	entrées d'index par page	<p>Nombre (fixe ou moyen ou minimal ou maximal) d'entrées d'index au plus par page : ce paramètres dépend de la structure de l'index $I(R, c)$.</p>
$nb_{pagesEntrees}^{fixe moy min max}(I(R, c))$	pages	<p>Nombre (fixe ou moyen ou minimal ou maximal) de pages nécessaires au stockage des entrées d'index (on ne compte donc PAS ici les pages stockant les informations de circulation dans l'index) :</p> $nb_{pagesEntrees}^{fixe moy min max}(I(R, c)) \stackrel{\text{Card}(R)}{=} \frac{nb_{entrees}^{exact}(I(R, c))}{nbMax_{entrees/page}^{fixe moy min max}(I(R, c))}$

Tableau 35. Performances d'indexation : paramètres généraux (indépendants de sa structure) de l'index

8.2 Structure d'un index

Il existe ainsi plusieurs structures d'index. Celles-ci diffèrent notamment par :

- La forme que prennent les informations de circulation dans l'index, i.e. la façon dont elles « organisent » les pages de l'index : on distingue notamment :
 - *Les structures arborescentes* : les informations de circulation organisent les pages de l'index en arborescence,
 - *Les structures à accès par hachage* : les informations de circulation organisent les pages de l'index en fonction d'une partition de hachage.
- Le type du constituant indexé, avec notamment :
 - *Les index à accès mono-critère* : le constituant indexé ne contient qu'un seul attribut,
 - *Les index à accès multi-critères* : le constituant indexé contient plusieurs attributs.



Remarque

Nous n'allons donc étudier que des structures d'index mono-critères (mais il y a déjà assez de choses à dire et montrer 😊).



En pratique

Les structures d'index que l'on rencontre le plus couramment sont les suivantes :

Type du constituant indexé c	Forme des informations de circulation	
	Structure arborescente (les informations de circulation organisent les pages de l'index en arborescence)	Structure à accès par hachage (les informations de circulation organisent les pages de l'index selon un hachage)
Index à accès mono-critère (c contient qu'un seul attribut)	<ul style="list-style-type: none"> ● Séquentiel-indexé ● Arbres B/B+ ● ... 	<ul style="list-style-type: none"> ● Statique (avec ou sans répertoire) ● Dynamique ● ...
Index à accès multi-critères (c contient plusieurs attributs)	<ul style="list-style-type: none"> ● Arbres k-d ● Arbres R ● ... 	<ul style="list-style-type: none"> ● Hachage partitionné ● ...

Tableau 36. Structures courantes d'index

Nous allons illustrer 3 de ces structures courantes d'index (en rouge ci-dessus) :

- Les arbres B+,
- Les index mono-critère à accès statique (les 2 variantes : avec et sans répertoire),
- Les index mono-critère à accès dynamique.



Données

Nous illustrons la présentation de ces 3 exemples de structures d'index par l'indexation primaire d'une relation Dictionnaire constituée de 2 attributs.

Dictionnaire	
<u>Mot</u> ¹³³	<u>Définition</u> ¹³⁴
bateau	Nom des embarcations, des navires autres que les navires de guerre
corde	Assemblage de fils tressés ou tordus ensemble
dessin	Représentation sur une surface d'une forme d'un objet ou d'une figure
ecole	Établissement où se donne un enseignement collectif
kayak	Embarcation étanche et légère, manœuvrée à la pagaie double
melodie	Suite de son formant un air
nez	Partie saillante du visage, entre la bouche et le front
terre	Planète habitée par l'homme
zébu	Bœuf à longues cornes et à bosse sur le garrot

Tableau 37. Relation Dictionnaire illustrant les structures d'index présentées

8.2.1 Arbres B+

Les arbres B ont été introduits par Bayer et Mc Creight en 1972 et ont fait l'objet de nombreux développements par la suite. Nous décrivons une variante des arbres B : les arbres B+. Ceux-ci sont en général utilisés dans les BD précisément comme outil d'indexation mono-critère des données¹³⁵.

8.2.1.1 Organisation générale

Comme leur nom l'indique, les arbres B+ sont... des arbres !!! 😊 Leurs nœuds sont des pages et ces pages sont « organisées en niveaux », chaque page d'un niveau « désignant » une ou plusieurs pages du niveau immédiatement « inférieur »¹³⁶ (sauf les feuilles, bien sûr, qui n'ont, par définition, pas de niveau « inférieur »). Dans les arbres B+, les pages peuvent être réparties en 2 « catégories » :

- D'une part, celles qui contiennent des informations de circulation dans l'index,
- D'autre part, celles qui contiennent (quasi-uniquement) des entrées d'index.



Attention

C'est vraiment soit l'un soit l'autre : on ne peut PAS avoir, dans un arbre B/B+, de pages qui contiennent à la fois des informations de circulation dans l'index et des entrées d'index.

¹³³ Par souci de simplicité pour la suite, on s'est « débarrassé » des accents dans l'écriture des mots du dictionnaire...

¹³⁴ Extraites d'un (très) vieux Larousse de Poche.

¹³⁵ Mais ils peuvent avoir d'autres usages dans d'autres domaines.

¹³⁶ La notion de « inférieur » peut dépendre du sens dans lequel vous dessinez l'arbre. C'est assez contre-intuitif mais les arbres B+ sont usuellement dessinés avec la racine en haut et les feuilles en bas. C'est en adoptant ce sens-là qu'on dit qu'une page désigne une ou plusieurs pages du niveau immédiatement inférieur.

Exemples

Pour vous donner une première idée de la structure de tels arbres, voici ci-dessous un arbre B¹³⁷ :

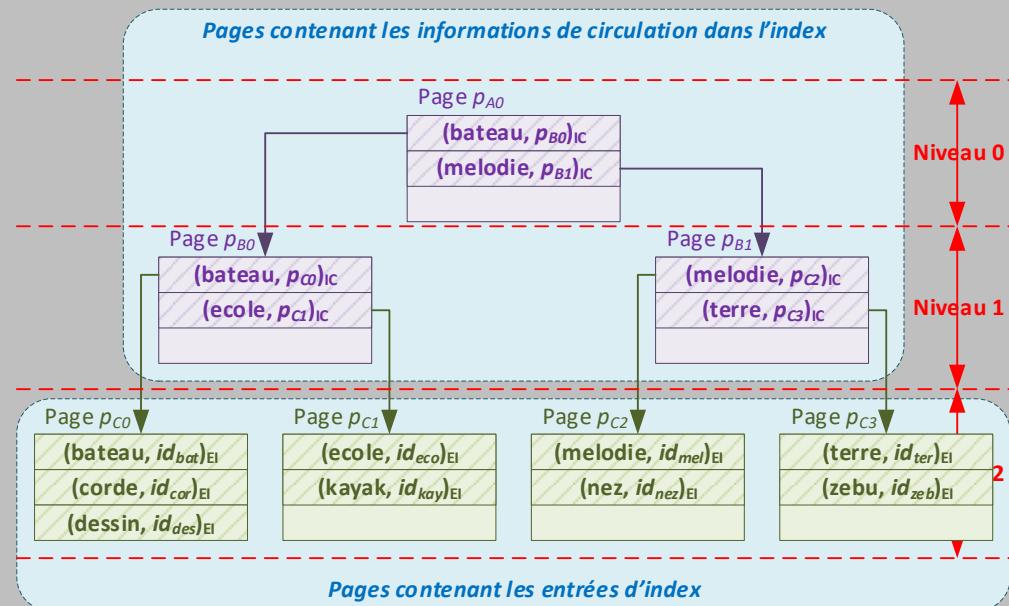


Figure 113. Exemple d'arbre B



Ci-dessous le même exemple d'index mais structuré en arbre B+ :

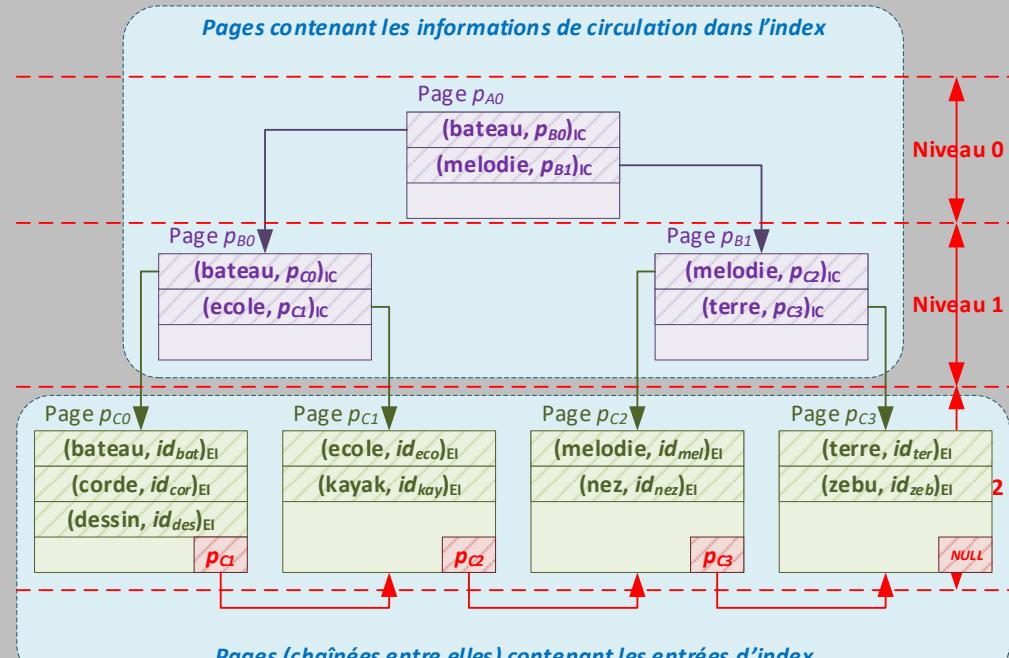


Figure 114. Exemple d'arbre B+

¹³⁷ Dans ces arbres, chaque « p_{A0} » est un identificateur de page et chaque « id_{xxx} » est une adresse logique d'un n-uplet (donc de la forme (p, i) en mode d'adressage direct ou id en mode d'adressage indirect).

Les exemples précédents donnent intuitivement de bonnes indications relatives aux propriétés des arbres B+. On peut, à la vision des 2 figures, raisonnablement penser :

- Que les arbres B+ sont des arbres binaires,
- Qu'ils semblent équilibrés,
- Qu'il semble y avoir un ordre à la fois au sein d'un même nœud mais aussi entre nœuds situés à un même niveau,
- Que la différence entre un arbre B et un arbre B+ semble être au niveau du chaînage des feuilles (chaînage simple), présent dans un arbre B+ mais pas dans un arbre B...



Question

Va pour les « intuitions », mais qu'en est-il en réalité ?

Réponse

La réalité n'est pas loin : elle est juste un peu plus précise que cela...

Précisément, ces arbres vérifient les contraintes suivantes :

- Les arbres B+ sont des arbres n-aires,



Rappel

Un arbre n-aire est un arbre dont chaque nœud peut désigner au niveau inférieur un nombre de nœuds, non limité, qui lui est propre.

- Ce sont des arbres parfaitement équilibrés,



Rappel

Un arbre est « équilibré » quand la profondeur de ses feuilles est la même à 1 près.
Un arbre est « parfaitement équilibré » quand la profondeur de ses feuilles est exactement la même.

- Ce sont des arbres de recherche, donc ordonnés (ici, entre nœuds d'un même niveau mais aussi au sein de chaque nœud : les enregistrements d'index contenus dans un même nœud sont ordonnés au sein de celui-ci),



Rappel

Un arbre est un « arbre de recherche » quand ses nœuds sont ordonnés selon une relation d'ordre qui lui est propre (ici, les nœuds d'un même niveau sont ordonnés par ordre croissant de la valeur du premier enregistrement d'index, donc celui « de plus petite valeur », qu'ils contiennent).

- Effectivement, il y a dans un arbre B+ un chaînage simple entre les feuilles, chaînage qui n'existe pas dans un arbre B.



Remarque

Chaque feuille est en effet chaînée vers la feuille immédiatement à sa droite (sauf la dernière feuille qui, forcément, n'a pas de sœur à sa droite). Ce chaînage respecte donc la relation d'ordre entre les feuilles.



Définition : « arbre B+ »

Un **arbre B+** est, dans le cadre des SGBD, une structure d'index. C'est donc une structure prenant la forme d'un arbre de recherche (inter et intra-nœuds) n'aire parfaitement équilibré. Les nœuds de cet arbre sont des pages et contiennent :

- Pour les nœuds terminaux¹³⁸ : quasi-uniquement des entrées d'index,
- Pour les nœuds non-terminaux¹³⁹ : des informations de circulation dans l'index.

Dans un arbre B+, les feuilles sont chaînées entre elles suivant la relation d'ordre.

Enfin, un arbre B+ est défini par une unique caractéristique que l'on appelle son **ordre**. Celui-ci détermine le nombre d'enregistrements d'index que l'on peut mettre au maximum dans ses nœuds.



Définition : « ordre d'un arbre B+ »

L'**ordre d'un arbre B+** est l'unique caractéristique de cette forme d'index et se note $e_{max}(I(R, c))$ pour un index $I(R, c)$. Cet ordre détermine le « comportement » de l'arbre tout au long de sa vie. La valeur de l'ordre est uniforme sur tout l'arbre (donc la même quel que soit le nœud considéré dans l'arbre). Cette caractéristique est valuée par un nombre entier **impair** supérieur ou égal à 3^{140,141} ($e_{max}(I(R, c)) \geq 3$).

8.2.1.2 Organisation détaillée

L'ordre d'un arbre B+ indique notamment les « remplissages » (i.e. le nombre d'enregistrements d'index) minimal et maximal de ses nœuds. Il dicte donc la façon dont l'arbre évolue au cours du temps.

8.2.1.2.1 Nœuds terminaux (hors racine si elle est seule) et entrées d'index

Les feuilles d'un arbre B+, on l'a dit, contiennent quasi-exclusivement des entrées d'index.



Rappel

Une entrée d'index est toujours un couple de la forme (v, a) où v est une valeur du constituant indexé et a est une liste d'adresses logiques : celles de tous les n-uplets de la relation indexée pour lesquels le constituant indexé vaut v .

Pour être précis, une feuille contient un nombre (borné) d'entrées d'index ET, à la fin de la page, une « pointeur » (i.e. un identificateur de page) vers la feuille suivante (i.e. immédiatement à droite).

¹³⁸ Rappelons qu'un nœud terminal, dans le cadre d'un arbre, est une feuille.

¹³⁹ Rappelons qu'un nœud non-terminal, dans le cadre d'un arbre, est un nœud qui n'est pas une feuille.

¹⁴⁰ Vous comprendrez pourquoi plus loin.

¹⁴¹ Dans certaines présentations des arbres B+, l'ordre de l'arbre est simplement un entier strictement positif : son usage est alors légèrement différent mais, finalement, cela revient exactement au même.



En pratique

Par construction, le nombre d'entrées d'index contenues dans une feuille d'un arbre B+ est TOUJOURS¹⁴² compris dans un intervalle donné dont les bornes dépendent de l'ordre $e_{max}(I(R, c))$ de cet arbre :

$$\frac{e_{max}(I(R, c)) + 1}{2} \leq nb_{entrees/feuille}^{exact}(I) \leq e_{max}(I(R, c))$$

Équation 33. Remplissage des feuilles (hors racine) d'un arbre B+



Exemple

Soit un arbre B+ d'ordre $e_{max}(I(R, c)) = 3$. Les feuilles de l'arbre contiennent donc toujours entre 2 et 3 entrées d'index (plus le « pointeur » vers leur sœur).

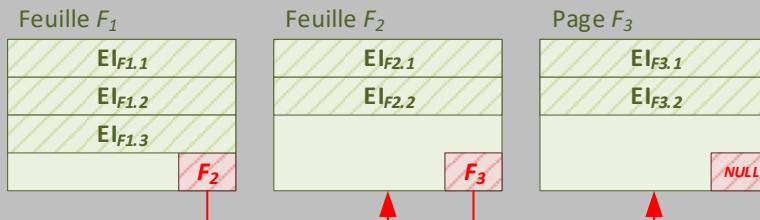


Figure 115. Exemple de feuilles d'un arbre B+ dont l'ordre $e_{max}(I(R, c))$ vaut 3

Enfin, les entrées d'index stockées dans une même feuille sont toujours stockées dans cette feuille par ordre croissant de leur valeur v du constituant indexé.



Remarque

Une entrée d'index n'est JAMAIS stockée « à cheval » sur 2 feuilles¹⁴³.

8.2.1.2.2 Nœuds non-terminaux (hors racine) et informations de circulation

Les nœuds non-terminaux (**sauf la racine**) contiennent exclusivement des informations de circulation dans l'index. Dans le cas des arbres B+¹⁴⁴, une information de circulation dans l'index est un couple de la forme (v, p) où v est une valeur du constituant indexé et p est un « pointeur » (i.e. un identificateur de page) vers une page située au niveau immédiatement inférieur dans l'arbre.

¹⁴² Sinon, c'est que l'arbre a mal été construit.

¹⁴³ Il y a en réalité des cas particuliers (notamment si la liste a d'adresses logiques associée à une ou plusieurs entrées d'index est longue) mais pas dans le sens où on pourrait l'entendre ici.

¹⁴⁴ Et uniquement pour cette structure d'index-là.



Remarque

Vous aurez noté l'homogénéité d'expression des enregistrements d'index dans le cadre des arbres B+ : ce sont toujours des couples de la forme (v , « quelque chose ») où v est une valeur du constituant indexé. Plus précisément :

- Pour une entrée d'index associée à une valeur v du constituant indexé : le « quelque chose » est une liste a des adresses logiques de tous les n-uplets de la relation indexée possédant cette valeur v pour le constituant indexé,
- Pour une information de circulation dans l'index associée à une valeur v du constituant indexé : le « quelque chose » est un « pointeur » (i.e. un identificateur de page) vers une page du niveau immédiatement inférieur.

Pour être plus précis, le nombre d'informations de circulation dans l'index que l'on peut trouver dans un nœud non-terminal (sauf la racine) est, lui aussi, borné.



En pratique

Le nombre d'informations de circulation dans l'index contenues dans un nœud non-terminal (sauf la racine) d'un arbre B+ est TOUJOURS¹⁴⁵ compris dans un intervalle donné dont les bornes dépendent de l'ordre $e_{max}(I(R, c))$ de cet arbre :

$$\frac{e_{max}(I(R, c)) + 1}{2} \leq nb_{infos/noeudNT}^{exact}(I) \leq e_{max}(I(R, c))$$

Équation 34. Remplissage des nœuds non-terminaux (hors racine) d'un arbre B+

Ainsi, un arbre B+ d'ordre $e_{max}(I(R, c)) = 3$ a toujours, dans ses nœuds non-terminaux (sauf la racine) 2 ou 3 informations de circulation dans l'index.

De plus, les informations de circulation dans l'index stockées dans un même nœud non-terminal (sauf la racine) sont toujours stockées dans ce nœud par ordre croissant de leur valeur v du constituant indexé.

Enfin, par construction, la page pointée par une information de circulation dans l'index associée à une valeur v du constituant indexé contient uniquement des enregistrements d'index associés à une valeur supérieure ou égale à v et la plus petite de ces valeurs est forcément exactement la valeur v (rappelons que, au sein de cette page, ces enregistrements d'index sont triés par ordre croissant de ces valeurs).



Remarque

Une information de circulation dans l'index n'est JAMAIS stockée « à cheval » sur 2 nœuds non-terminaux¹⁴⁶.

¹⁴⁵ Sinon, c'est que l'arbre a mal été construit.

¹⁴⁶ Et il n'y a *a priori* pas d'exceptions ici.

Remarque

Ainsi, supposons que, dans un nœud non-terminal (et non-racine) d'un arbre B+, on a la séquence, ordonnée donc, d'informations de circulation dans l'index suivante¹⁴⁷ : $(v_1, p_1), (v_2, p_2), \dots, (v_m, p_m)$. On peut alors dire, par construction (cf. Figure 116) :

- Pour i allant de 1 à $m - 1$, p_i est un pointeur vers la racine du sous-arbre dont les enregistrements d'index sont associés à une valeur égale ou supérieure à v_i et inférieure strictement à v_{i+1} (i.e. $v_i \leq v < v_{i+1}$).
- p_m est un pointeur vers la racine du sous-arbre dont les enregistrements d'index sont associés à une valeur égale ou supérieure à v_m (i.e. $v_m \leq v$).

Notons enfin que La valeur v_1 est parfois « virtuelle » : elle n'est parfois pas stockée (mais uniquement elle, on stocke quand même l'identificateur de page p_1 associé !) car elle n'est pas nécessaire pour la recherche d'une entrée d'index (implicitement, elle pointe vers la racine du sous-arbre dont les feuilles contiennent les entrées d'index dont la valeur est inférieure strictement à v_2). Ceci est logique puisque, toujours par construction, v_1 est la plus petite valeur possible du constituant indexé que l'on peut atteindre depuis le nœud courant est descendant dans l'arbre B+. **Cependant, pour éviter des erreurs fréquentes, nous ne nous permettrons PAS de sous-entendre la valeur v_1 de chaque nœud.**

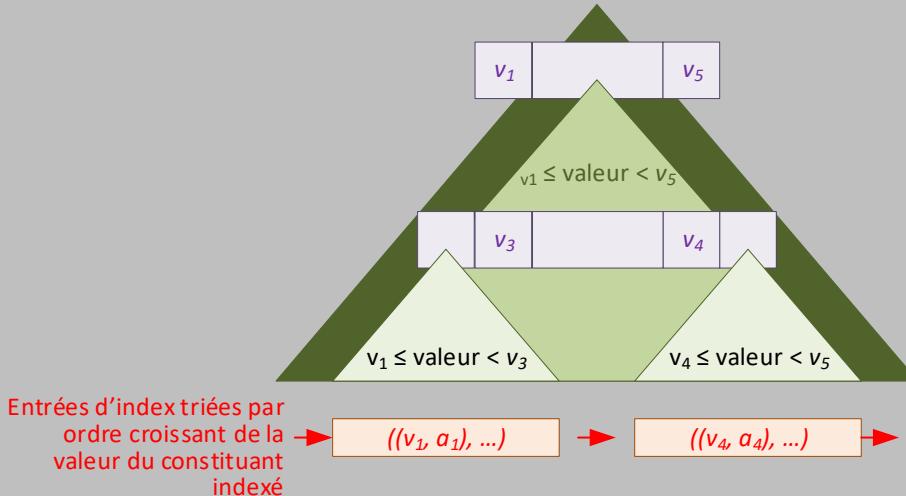


Figure 116. Allure générale d'un arbre B+

¹⁴⁷ Donc avec $v_1 \leq v_2 \leq \dots \leq v_i \leq \dots \leq v_m$.

8.2.1.2.3 Racine et enregistrements d'index

La racine d'un arbre B+ est un cas particulier :

- *S'il existe peu de valeurs différentes dans la relation du constituant indexé* : il existe alors dans l'index peu d'entrées d'index. Il se peut donc fort bien que l'arbre B+ ne contienne alors qu'une seule page, donc uniquement la racine, qui est alors l'unique nœud de l'arbre.
- *S'il existe un nombre raisonnable ou grand de valeurs différentes dans la relation du constituant indexé* : il existe alors dans l'index pas mal voire beaucoup d'entrées d'index. Dans ce cas, l'arbre B+ contient plusieurs nœuds (une racine, éventuellement d'autres nœuds non-terminaux, et enfin des feuilles).

On distingue donc pour la racine les 2 cas suivants :

- *Quand la racine est l'unique nœud de l'arbre B+* : c'est aussi une feuille (la seule !) de l'arbre, elle contient des entrées d'index (ainsi que la place nécessaire pour stocker un identifiant de page à sa fin, tout comme les autres feuilles non-racines quand il y a plus d'un seul niveau dans l'arbre),
- *Quand la racine n'est PAS l'unique nœud de l'arbre B+* : elle contient uniquement des informations de circulation dans l'index.

Dans les 2 cas, le nombre d'enregistrements d'index que l'on peut trouver dans la racine est, là aussi, borné (mais les bornes à prendre en compte sont différentes dans chacun de ces 2 cas).

En pratique

Le bornage du nombre d'enregistrements d'index que l'on peut trouver dans la racine d'un arbre B+ est le suivant :

- *Si la racine est le seul nœud de l'arbre B+* : elle contient un nombre borné d'entrées d'index

$$1 \leq nb_{entrees/racine}^{exact}(I) \leq e_{max}(I(R, c))$$

Équation 35. Remplissage de la racine d'un arbre B+ (quand elle est son seul nœud)



- *Si la racine n'est PAS le seul nœud de l'arbre B+* : elle contient un nombre borné d'informations de circulation dans l'index

$$2 \leq nb_{infos/racine}^{exact}(I) \leq e_{max}(I(R, c))$$

Équation 36. Remplissage de la racine d'un arbre B+ (quand elle n'est pas son seul nœud)

Enfin, les enregistrements d'index stockées dans la racine sont toujours stockés dans cette page « particulière » par ordre croissant de leur valeur v du constituant indexé.

8.2.1.2.4 Pages d'un arbre B+ et de la relation indexée

Du coup, on peut globalement voir comme suit les différentes structures de pages intervenant dans le stockage d'un index en arbre B+ et dans celui de la relation indexée par cet arbre B+ :

- Si la relation indexée est en mode d'adressage direct :

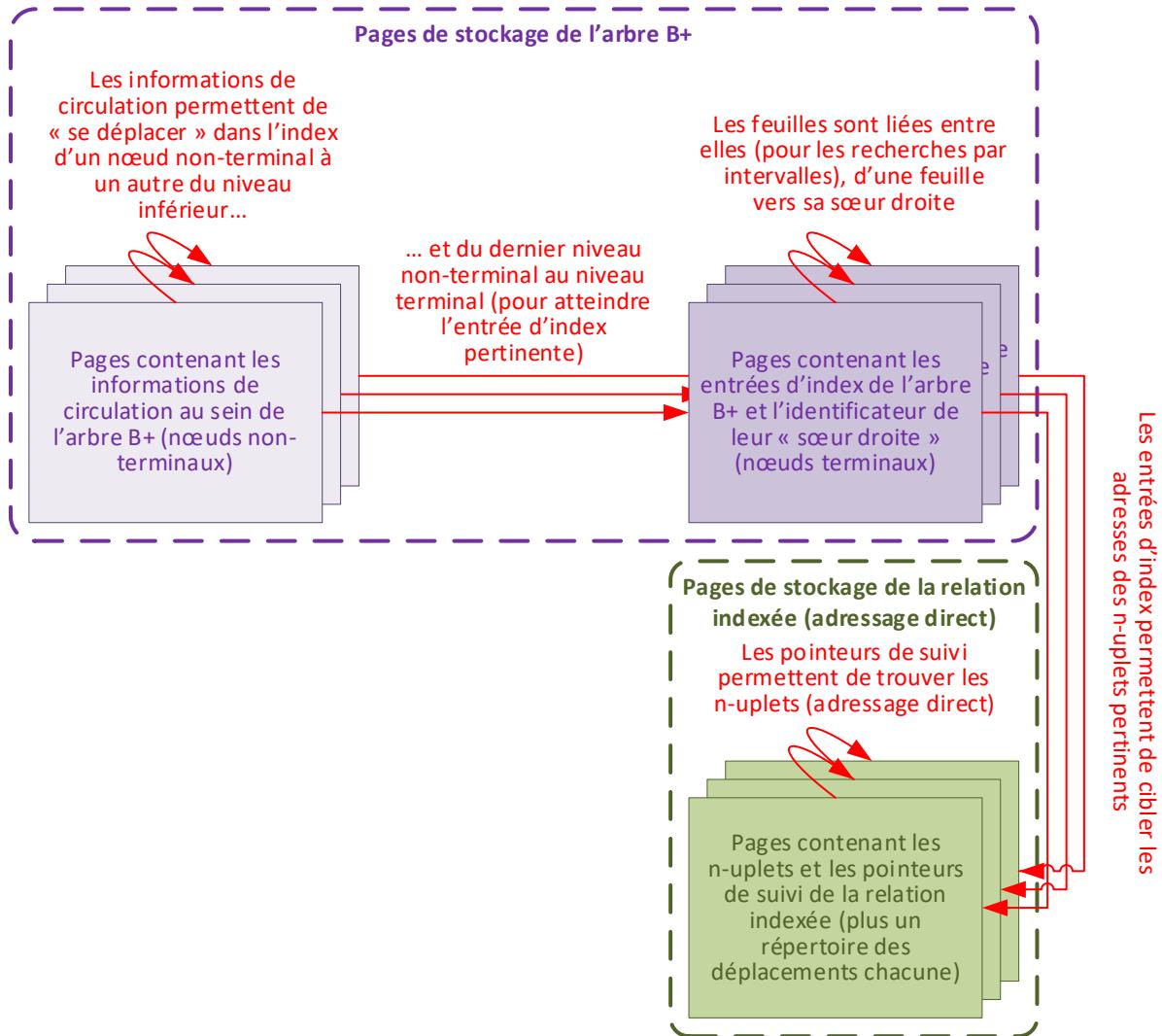


Figure 117. Pages de stockage d'un arbre B+ et de la relation indexée (adressage direct)

- Si la relation indexée est en mode d'adressage indirect :

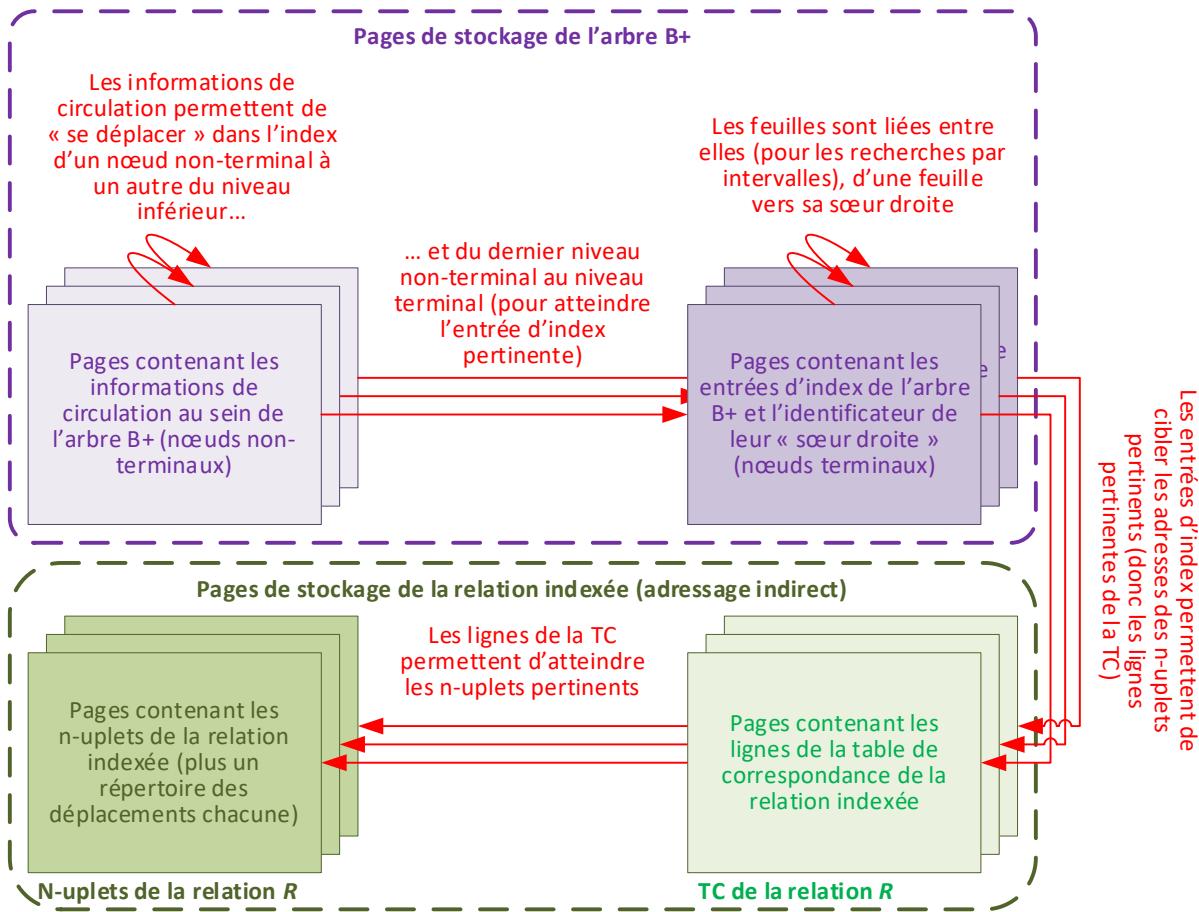


Figure 118. Pages de stockage d'un arbre B+ et de la relation indexée (adressage indirect)

8.2.1.3 Évolution d'un arbre B+ : opérations de manipulation

On l'a déjà dit, un index (donc un arbre B+) vit avec la relation qu'il indexe (cf. §8.1.4) :

- Quand cette relation ne contient aucun n-uplet : l'index est vide (aucune page),
- Quand il existe dans la relation un nombre $n \leq e_{\max}(l(R, c))$ de valeurs différentes du constituant indexé : l'arbre est formé seulement de sa racine, qui contient des entrées d'index,
- Quand il existe dans la relation un nombre $n > e_{\max}(l(R, c))$ de valeurs différentes du constituant indexé : l'arbre est constitué de plusieurs pages, certaines étant des nœuds non-terminaux (racine incluse) contenant des informations de circulation dans l'index et d'autres étant des nœuds terminaux contenant des entrées d'index.

On l'a vu aussi, les ajouts/modifications/suppressions de n-uplets vont entraîner des ajouts/suppressions/modifications d'entrées d'index (cf. §8.1.4). Conséquemment, cela va aussi pouvoir entraîner des ajouts/modifications/suppressions d'informations de circulation dans l'index.



Question

Le nombre d'enregistrements d'index que l'on peut mettre dans un nœud d'un arbre B+ étant borné en fonction de son ordre $e_{max}(I(R, c))$, comment respecter ces bornes au fur et à mesure des ajouts et suppressions ?

Réponse

Par construction, en réalisant les manipulations adéquates quand ces bornes sont atteintes.

Les opérations usuellement réalisées sur un arbre B+ sont au nombre de 5...

8.2.1.3.1 Recherche d'une entrée d'index

La recherche d'une entrée d'index (v, a) se fait d'après la valeur v : on recherche alors l'entrée d'index associée à la valeur v du constituant indexé (s'il en existe une, sinon la recherche ne retourne rien).

Globalement, l'idée est la suivante :

- Si on est sur un nœud de l'arbre contenant des informations de circulation dans l'index (nœud non-terminal ou racine quand elle n'est pas seule) : on compare la valeur v cherchée avec les valeurs associées aux informations de circulation dans l'index contenues dans le nœud. En fonction du résultat de ces comparaisons, on « suit » le pointeur contenu dans l'information de circulation adéquate vers une page du niveau immédiatement inférieur.



Rappel

Un tel nœud contient une liste ordonnée d'informations de circulation dans l'index de la forme suivante¹⁴⁸ : $(v_1, p_1), (v_2, p_2), \dots, (v_m, p_m)$. On peut alors dire :

- Pour i allant de 1 à $m - 1$, p_i est un pointeur vers la racine du sous-arbre dont les enregistrements d'index sont associés à une valeur égale ou supérieure à v_i et inférieure strictement à v_{i+1} .
- p_m est un pointeur vers la racine du sous-arbre dont les enregistrements d'index sont associés à une valeur égale ou supérieure à v_m .

- Si on est sur un nœud de l'arbre contenant des entrées d'index (nœud terminal ou racine quand elle est seule) : si on est sur un tel nœud, c'est qu'on est sur celui qui contient l'entrée d'index recherchée, si elle existe (on est donc certain qu'elle n'est pas dans une autre feuille). On compare la valeur v cherchée avec les valeurs associées aux entrées d'index contenues dans le nœud. Si on trouve une entrée d'index associée à une valeur identique, on la récupère (en fait, on récupère la liste a d'adresses logiques qu'elle contient). Sinon, c'est qu'il n'existe pas dans l'arbre B+ d'entrée d'index associée à la valeur v ciblée.

¹⁴⁸ Avec $v_1 \leq v_2 \leq \dots \leq v_i \leq \dots \leq v_m$.

En pratique

Le pseudo-algorithme suivant reflète le fonctionnement de la recherche.

```

Pseudo-algorithme RechercheEIB+
Données d'entrée
    v : une valeur
Données de sortie
    EI : l'entrée d'index recherchée
    p : le nœud de l'arbre B+ contenant EI
Données intermédiaires
    contenu : liste d'enregistrements d'index
Début
    p ← nœud racine de l'arbre B+
    Tant que p n'est pas un nœud non-terminal faire
        contenu ← ((v1, p1), (v2, p2), ..., (vn, pn)) de p
        Si v < v2 alors
            p ← p1
        Sinon
            Chercher dans contenu le premier vi ≤ v
            p ← pi (associé au vi trouvé ci-dessus)
        FinSi
    FinTQ
    Chercher dans p l'entrée d'index de valeur v
    Si elle existe alors
        EI ← entrée d'index (v, a) trouvée ci-dessus
    Sinon
        EI ← « EI INEXISTANTE => n-uplet inexistant »
    FinSi
Fin
```



8.2.1.3.2 Insertion d'une entrée d'index

L'insertion d'une entrée d'index (v, a) se fait d'après la valeur v : on recherche dans quel nœud terminal placer cette entrée d'index associée à la valeur v du constituant indexé et on l'insère (en respectant le classement par ordre croissant des entrées d'index contenues dans cette feuille).

L'idée générale est la suivante : la feuille dans laquelle insérer l'entrée d'index (v, a) est celle qui la contiendrait si elle existait déjà. S'il existe déjà dans cette feuille une entrée d'index associée à la valeur v, on n'insère alors pas de nouvelle entrée d'index mais on complète la liste d'adresses logiques qu'elle contient avec la liste d'adresses logiques a contenue dans l'entrée d'index à insérer. Sinon on réalise l'insertion dans la page, quitte à devoir la diviser en 2 si elle venait à être « trop pleine ».

En pratique

Le pseudo-algorithme suivant illustre l'insertion d'une entrée d'index.

Pseudo-algorithme InsertionEIB+

Données d'entrée

EI_{ins} : l'entrée d'index (v, a) à insérer

Données de sortie

p : page dans laquelle l'insertion est faite

Données intermédiaires

EI_{ch} : une entrée d'index

Début

Rechercher une entrée d'index EI_{ch} de valeur v^{149}

$EI_{ch} \leftarrow$ entrée d'index trouvée

$p \leftarrow$ nœud final de la recherche

Si EI_{ch} existe **alors**

Rajouter a aux adresses logiques contenues dans EI_{ch}

Sinon

Si index vide **alors**

$p \leftarrow$ nouvelle page

FinSi

Si $e_{max}(I(R, c)) < (Nb_{EI}(p) + 1)$ **alors**

Diviser la page p^{150}

$p \leftarrow$ page adéquate issue de la division

FinSi

Insérer EI_{ins} dans p en respectant l'ordre croissant

FinSi

Fin



8.2.1.3.3 Insertion d'une information de circulation dans l'index

Une telle insertion ressemble à celle d'une entrée d'index¹⁵¹ (mais elle reste plus simple que celle-ci) si ce n'est qu'on n'insère pas une donnée de même nature, que l'insertion est réalisée dans un nœud non-terminal et que l'on indique déjà en entrée dans quelle page cette insertion est à réaliser.



Attention

L'opération d'insertion d'une information de circulation ne peut PAS être déclenchée à la demande : elle ne peut être déclenchée que par la mise en œuvre d'une autre opération (ici, par l'opération d'insertion d'une entrée d'index).

¹⁴⁹ Voir le pseudo-algorithme RechercheEIB+ plus haut (cf. §8.2.1.3.1).

¹⁵⁰ Voir le pseudo-algorithme DivisionPageB+ plus bas (cf. §8.2.1.3.4).

¹⁵¹ Voir le pseudo-algorithme InsertionEIB+ plus haut (cf. §8.2.1.3.2).

En pratique

Le pseudo-algorithme suivant illustre l'insertion d'une information de circulation dans l'index.

Pseudo-algorithme InsertionICB+

Données d'entrée

IC : l'information de circulation (v, p) à insérer
 p : page dans laquelle l'insertion est à réaliser

Données de sortie

p (voir données d'entrée)

Données intermédiaires

Aucune

Début

Si $e_{max}(I(R, c)) < (Nb_{IC}(p) + 1)$ **alors**

Diviser la page p ¹⁵²
 $p \leftarrow$ page adéquate issue de la division

FinSi

Insérer IC dans p en respectant l'ordre croissant

Fin



8.2.1.3.4 Division d'une page d'un arbre B+

La division d'une page est réalisée quand un nœud devient « trop plein » (i.e. qu'il contient un nombre d'enregistrements d'index strictement supérieur à l'ordre $e_{max}(I(R, c))$ de l'arbre B+)...



Attention

L'opération de division d'une page ne peut PAS être déclenchée à la demande : elle ne peut être déclenchée que par la mise en œuvre d'une autre opération (ici, par une des opérations d'insertion présentées plus haut).

Les règles de remplissage des nœuds d'un arbre B+ imposent en effet un nombre maximum d'enregistrements d'index dans un nœud égal à l'ordre $e_{max}(I(R, c))$ de l'arbre B+. Ainsi, quand on insère des enregistrements d'index, ce nombre peut assez souvent être dépassé (par le haut). On divise alors la page dans laquelle l'insertion doit être réalisée en 2 et l'insertion se fait alors dans une des pages issues de cette division. Enfin, cette division fait « remonter » des enregistrements d'index (à insérer au niveau immédiatement supérieur, ce qui peut à nouveau provoquer des divisions, voire des fusions).

¹⁵² Voir le pseudo-algorithme DivisionPageB+ plus bas (cf. §8.2.1.3.4).

En pratique

La division d'un page est 2 est décrite par le pseudo-algorithme suivant.

Pseudo-algorithme DivisionPageB+

Données d'entrée

p : la page à diviser

Données de sortie

p (voir données d'entrée) : 1^{ère} page issue de la div.

p' : 2^{nde} page issue de la division

Données intermédiaires

contenu : une liste d'enregistrements d'index

contenu' : une liste d'enregistrements d'index

idSœur : un identificateur de page

enrInd : un enregistrement d'index

IC : une information de circulation dans l'index

racine : une éventuelle nouvelle racine de l'arbre B+

Début

$p' \leftarrow$ nouvelle page vide

contenu \leftarrow liste d'enregistrements d'index de p

contenu' \leftarrow 2^{nde} moitié de *contenu*

contenu \leftarrow 1^{ère} moitié de *contenu*

Écrire *contenu* dans p (écraser l'ancien *contenu*)

Écrire *contenu'* dans p'

Si p est la racine de l'arbre B+ **alors**

racine \leftarrow nouvelle page vide

enrInd \leftarrow 1^{er} enregistrement d'index (v , xxx) de p

IC \leftarrow nouvelle information de circulation (v , p)

Insérer *IC* dans *racine*¹⁵³

Sinon

Si p est une feuille (non racine) **alors**

idSœur \leftarrow identifiant de la sœur droite de p

Écrire *idSœur* à la fin de p'

Écrire l'identifiant de p' à la fin de p

FinSi

FinSi

enrInd \leftarrow 1^{er} enregistrement d'index (v , xxx) de p'

IC \leftarrow nouvelle information de circulation (v , p')

Insérer *IC* dans le nœud père de p ¹⁵³

Fin



¹⁵³ Voir le pseudo-algorithme InsertionICB+ plus haut (cf. §8.2.1.3.3).

8.2.1.3.5 Suppression d'une entrée d'index

La suppression d'une entrée d'index (v, a) se fait comme suit : on recherche dans quel nœud terminal se trouve cette entrée d'index et on la supprime (en fusionnant le nœud la contenant si besoin).

En pratique

Le pseudo-algorithme suivant illustre la suppression d'une entrée d'index.

```

Pseudo-algorithme SuppressionEIB+
Données d'entrée
    v : une valeur du constituant indexé
Données de sortie
    EI : l'entrée d'index à supprimer
    p : la page de laquelle EI est supprimée
Données intermédiaires
    EI' : une entrée d'index
    contenu : une liste d'entrées d'index
    IC : une information de circulation dans l'index
    IC' : une information de circulation dans l'index
    p' : une page de l'arbre B+
Début
    Rechercher l'entrée d'index de valeur v154
    Si elle n'existe pas alors
        p ← NULL
    Sinon
        EI ← entrée d'index (v, a) trouvée
        p ← nœud final de la recherche (contenant EI)
        contenu ← liste d'entrées d'index de p
        Si EI est la 1ère entrée d'index de contenu alors
            IC ← nouvelle info. de circulation (v, p)
            EI' ← entrée d'index (v', xxx) juste après EI
            IC' ← nouvelle info. de circulation (v', p)
        FinSi
        Supprimer EI de contenu
        Écrire contenu dans p (écrase l'ancien contenu)
        Si (NbEI(p) < ((emax(I(R, c))+1)/2))
            et (p non racine) alors
                p' ← Choisir une page avec laquelle fusionner p
                Fusionner p et p'155
                p ← page résultant de la fusion
        FinSi
        Si (IC et IC' définis) et (p non racine) alors
            Supprimer IC du père de p156
            Insérer IC' dans le père de p157
        FinSi
    FinSi
Fin

```



¹⁵⁴ Voir le pseudo-algorithme RechercheEIB+ plus haut (cf. §8.2.1.3.1).

¹⁵⁵ Voir le pseudo-algorithme FusionPagesB+ plus bas (cf. §8.2.1.3.7).

¹⁵⁶ Voir le pseudo-algorithme SuppressionICB+ plus bas (cf. §8.2.1.3.6).

¹⁵⁷ Voir le pseudo-algorithme InsertionICB+ plus haut (cf. §8.2.1.3.3).

L'idée générale est la suivante : la feuille de laquelle supprimer l'entrée d'index (v, a) est celle qui la contient, si elle existe. Si c'est le cas, on la supprime et on vérifie que cette feuille n'est pas « trop vide », sinon on la fusionne avec une autre feuille... Enfin, si l'entrée d'index supprimée était la première dans sa page (ET si cette page n'est pas la racine), on supprime au niveau immédiatement supérieur l'information de circulation de même valeur v et on insère à sa place l'information de circulation correspondant à la valeur d'entrée d'index immédiatement supérieure.

Question

Quelle feuille choisir pour la fusion avec la feuille courante ?

**Réponse**

On choisit forcément une feuille immédiatement à gauche ou à droite de la feuille courante. En cas d'indécision, on choisit de préférence celle qui a le même père que la feuille concernée. S'il reste encore une indécision, on choisit de préférence de réaliser la fusion qui n'entraînerait pas une division « dans la foulée ». Enfin, encore en cas d'indécision, on choisit de préférence celle qui est immédiatement à droite de la feuille courante.

8.2.1.3.6 Suppression d'une information de circulation dans l'index

Une telle suppression ressemble à celle d'une entrée d'index¹⁵⁸ (mais elle reste plus simple que celle-ci) si ce n'est qu'on ne supprime pas une donnée de même nature, que la suppression est réalisée dans un nœud non-terminal et que l'on indique déjà en entrée dans quelle page faire cette suppression.

**Attention**

L'opération de suppression d'une information de circulation ne peut PAS être déclenchée à la demande : elle ne peut être déclenchée que par la mise en œuvre d'une autre opération (ici, par l'opération de suppression d'une entrée d'index).

Globalement, le fonctionnement est le suivant : on supprime l'information de circulation dans l'index de la page donnée en entrée et on vérifie que cette page n'est maintenant pas « trop vide », sinon on la fusionne avec une autre page... De plus, si l'information de circulation dans l'index supprimée était la première dans sa page (et si cette page n'est pas la racine), on supprime au niveau immédiatement supérieur l'information de circulation de même valeur v et on insère à sa place l'information de circulation correspondant à la valeur associée à l'information de circulation dans l'index immédiatement supérieure.

Question

Quel nœud non-terminal choisir pour la fusion avec la page courante ?

**Réponse**

On choisit forcément un nœud immédiatement à gauche ou à droite de la page courante. Là encore, en cas d'indécision, on choisit de préférence le nœud qui a le même père que la page courante. S'il reste encore une indécision, on choisit de préférence de réaliser la fusion qui n'entraînerait pas une division « dans la foulée ». Enfin, encore en cas d'indécision, on choisit de préférence de fusionner avec le nœud qui est immédiatement à droite de la page courante.

¹⁵⁸ Voir le pseudo-algorithme InsertionEIB+ plus haut (cf. §8.2.1.3.2).

En pratique

Le pseudo-algorithme suivant illustre la suppression d'une entrée d'index.

Pseudo-algorithme SuppressionICB+

Données d'entrée

IC : l'information de circulation (v, p) à supprimer
 p_{IC} : la page de laquelle supprimer IC

Données de sortie

p_{IC} (voir données d'entrée)

Données intermédiaires

contenu : une liste d'entrées d'index

IC' : une information de circulation dans l'index

IC_{sup} : une information de circulation dans l'index

IC_{ins} : une information de circulation dans l'index

p' : une page de l'arbre B+

Début

$contenu \leftarrow$ liste d'info. de circulation de p_{IC}

Si IC est la 1^{ère} info. de circulation de $contenu$ **alors**

$IC_{sup} \leftarrow$ nouvelle info. de circulation (v, p_{IC})

$IC' \leftarrow$ info. de circulation (v', p') juste après IC

$IC_{ins} \leftarrow$ nouvelle info. de circulation (v', p_{IC})

FinSi

Supprimer IC de $contenu$

Écrire $contenu$ dans p (écrase l'ancien $contenu$)

Si ($Nb_{IC}(p) < ((e_{max}(I(R, c)) + 1)/2)$)

et (p non racine) **alors**

$p' \leftarrow$ Choisir une page avec laquelle fusionner p

$p \leftarrow$ Fusionner p et p' ¹⁵⁹

FinSi

Si (IC' et IC'' définis) **et** (p non racine) **alors**

Supprimer IC_{sup} du père de p ¹⁶⁰

Insérer IC_{ins} dans le père de p ¹⁶¹

FinSi

Si (père de p racine) **et** ($2 < Nb_{IC}(\text{père de } p)$) **alors**

Supprimer la racine (p devient racine, unique nœud)

FinSi

Fin



8.2.1.3.7 Fusion de 2 pages d'un arbre B+

La fusion de 2 pages est réalisée quand un nœud devient « trop vide » (i.e. qu'il contient un nombre d'enregistrements d'index strictement inférieur à $((e_{max}(I(R, c)) + 1)/2)$...



Attention

L'opération de fusion de 2 pages ne peut PAS être déclenchée à la demande : elle ne peut être déclenchée que par la mise en œuvre d'une autre opération (ici, par une des opérations de suppression présentées plus haut).

¹⁵⁹ Voir le pseudo-algorithme FusionPagesB+ plus bas (cf. §8.2.1.3.7).

¹⁶⁰ Appel récursif du même pseudo-algorithme SuppressionICB+.

¹⁶¹ Voir le pseudo-algorithme InsertionICB+ plus haut (cf. §8.2.1.3.3).

Les règles de remplissage des nœuds d'un arbre B+ imposent en effet un nombre minimum d'enregistrements d'index dans un nœud. En l'occurrence, ce nombre minimum est égal à $((e_{max}(I(R, c)) + 1)/2)$ où $e_{max}(I(R, c))$ est l'ordre de l'arbre B+. Ainsi, quand on supprime des enregistrements d'index, ce nombre peut assez souvent être dépassé (par le bas). Dans de tels cas, on fusionne la page dans laquelle la suppression a été réalisée avec une de ses sœurs immédiates. Enfin, cette fusion fait « remonter » des enregistrements d'index (à insérer au niveau immédiatement supérieur, ce qui peut à nouveau provoquer des fusions, voire des divisions).

En pratique

La fusion de 2 pages est décrite par le pseudo-algorithme suivant.

Pseudo-algorithme FusionPagesB+

Données d'entrée

p : la première page (celle de gauche) à fusionner
 p' : la seconde page (celle de droite) à fusionner

Données de sortie

p (voir données d'entrée) : résultat de la fusion

Données intermédiaires

$contenu$: une liste d'enregistrements d'index

$contenu'$: une liste d'enregistrements d'index

$idSœur$: un identificateur de page

EI : une entrée d'index

IC : une information de circulation

Début

$contenu \leftarrow$ liste d'enregistrements d'index de p

$contenu' \leftarrow$ liste d'enregistrements d'index de p'

$EI \leftarrow$ 1^{ère} entrée d'index (v, xxx) de $contenu'$

$IC \leftarrow$ nouvelle info. de circulation (v, p')

$contenu \leftarrow$ concaténation de $contenu$ et $contenu'$

Écrire $contenu$ dans p (écrase ancien $contenu$)

Si p et p' sont des feuilles **alors**

$idSœur \leftarrow$ pointeur vers la feuille suivante de p'

Remplacer pointeur vers feuille sœur de p par $idSœur$

FinSi

Supprimer IC du père de p' ¹⁶²

Supprimer p'

Si $e_{max}(I(R, c)) < Nb_{EI}(p)$ **alors**

Diviser la page p ¹⁶³

$p \leftarrow$ page gauche issue de la fusion

Finsi

Fin



¹⁶² Voir pseudo-algorithme SuppressionICB+ plus haut (cf. §8.2.1.3.6).

¹⁶³ Voir pseudo-algorithme DivisionPageB+ plus haut (cf. §8.2.1.3.4).

8.2.1.3.8 Exemple de manipulation d'un index en arbre B+ d'ordre 3

On se propose de montrer l'évolution d'un arbre B+ au fur et à mesure de l'ajout/suppression de n-uplets à/de la relation indexée *Dictionnaire* (cf. Tableau 37). On suppose que cette relation est initialement vide et que l'index que l'on considère est un index primaire ayant la structure d'un arbre B+ d'ordre $e_{max}(I(Dictionnaire, Mot)) = 3$. Avec cette valeur, on sait que :

- Toute feuille (sauf la racine si elle est seule) contient 2 ou 3 entrées d'index,
- Tout nœud non-terminal (sauf la racine) contient 2 ou 3 informations de circulation dans l'index,
- La racine, si elle est seule, contient 1 à 3 entrées d'index,
- La racine, si elle n'est pas seule, contient 2 ou 3 informations de circulation dans l'index.

Exemple (début)

La relation va évoluer opération après opération, l'index associé également :

1. Initialement, la relation *Dictionnaire* est vide, l'index aussi :

Le néant

2. On ajoute dans la relation le mot *melodie*, donc l'entrée d'index associée à la valeur *melodie* est insérée dans l'index : la première page F_0 de l'index est créée et l'entrée d'index voulue y est rajoutée.

Page F_0
(melodie, id_{mel}) _{EI}
NULL

3. On ajoute dans la relation le mot *ecole*, donc l'entrée d'index associée à la valeur *ecole* est insérée dans l'index : il y a de la place dans la page qui contiendrait l'entrée d'index à insérer, i.e. la page F_0 , donc on la rajoute dans cette page (en respectant l'ordonnancement par ordre croissant).

Page F_0
(ecole, id_{eco}) _{EI}
(melodie, id_{mel}) _{EI}
NULL

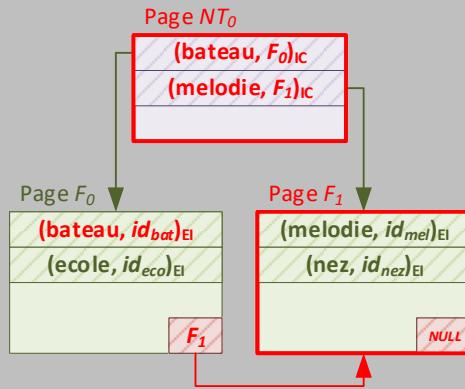
4. On ajoute dans la relation le mot *nez*, donc l'entrée d'index associée à la valeur *nez* est insérée dans l'index : là encore, il y a de la place dans la page qui contiendrait l'entrée d'index à insérer, i.e. la page F_0 , donc on la rajoute dans cette page (toujours en respectant l'ordonnancement).

Page F_0
(ecole, id_{eco}) _{EI}
(melodie, id_{mel}) _{EI}
(nez, id_{nez}) _{EI}
NULL

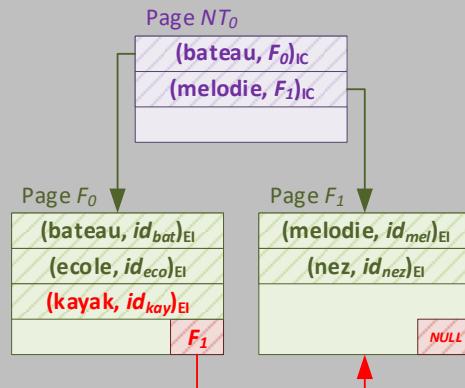


Exemple (suite)

5. On ajoute dans la relation le mot bateau, donc l'entrée d'index de valeur bateau est insérée dans l'index : si l'on rajoute cette entrée d'index dans l'unique page F_0 , celle-ci serait trop pleine. Elle est donc divisée en 2 pages, F_0 et F_1 , à l'insertion de l'entrée d'index. Cette division fait remonter des informations de circulation au niveau immédiatement supérieur, niveau qui n'existe pas. Il est donc créé (la hauteur de l'arbre augmente de 1), c'est la page NT_0 , et les informations de circulation adéquates peuvent être remontées.

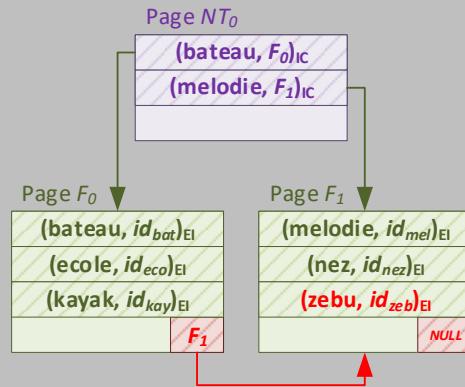


6. On ajoute dans la relation le mot kayak, donc l'entrée d'index de valeur kayak est insérée dans l'index : il y a de la place dans la page qui contiendrait l'entrée d'index à insérer, i.e. la page F_0 , donc on la rajoute dans cette page (toujours en respectant l'ordonnancement).

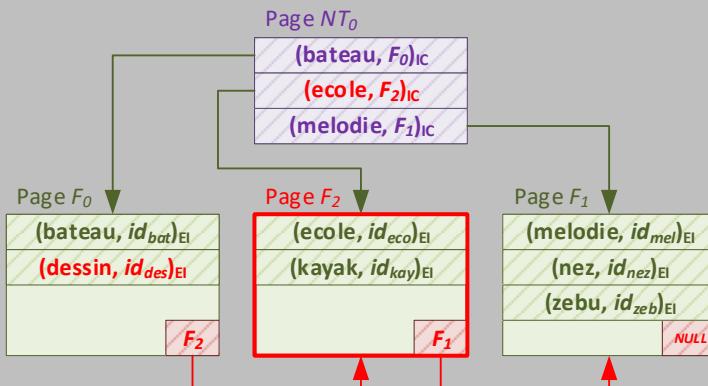


Exemple (suite)

7. On ajoute dans la relation le mot zebu, donc l'entrée d'index de valeur zebu est insérée dans l'index : il y a de la place dans la page qui contiendrait l'entrée d'index à insérer, *i.e.* la page F_1 , donc on la rajoute dans cette page (toujours en respectant l'ordonnancement).

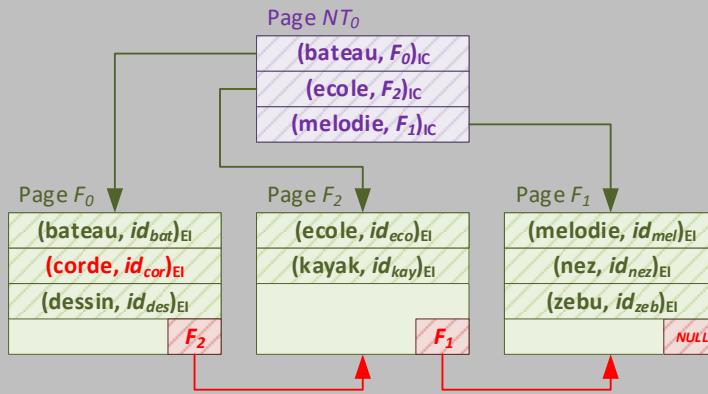


8. On ajoute dans la relation le mot dessin, donc l'entrée d'index de valeur dessin est insérée dans l'index : si l'on rajoute cette entrée d'index dans la page F_0 où elle devrait être insérée, celle-ci serait trop pleine. Elle est donc divisée en 2 pages, F_0 et F_2 , à l'insertion de l'entrée d'index. Cette division fait remonter des informations de circulation au niveau immédiatement supérieur, niveau qui ne contient que la page NT_0 . Pas de souci à ce niveau-là : la page NT_0 contient suffisamment de place libre pour qu'on y insère une nouvelle information de circulation dans l'index.

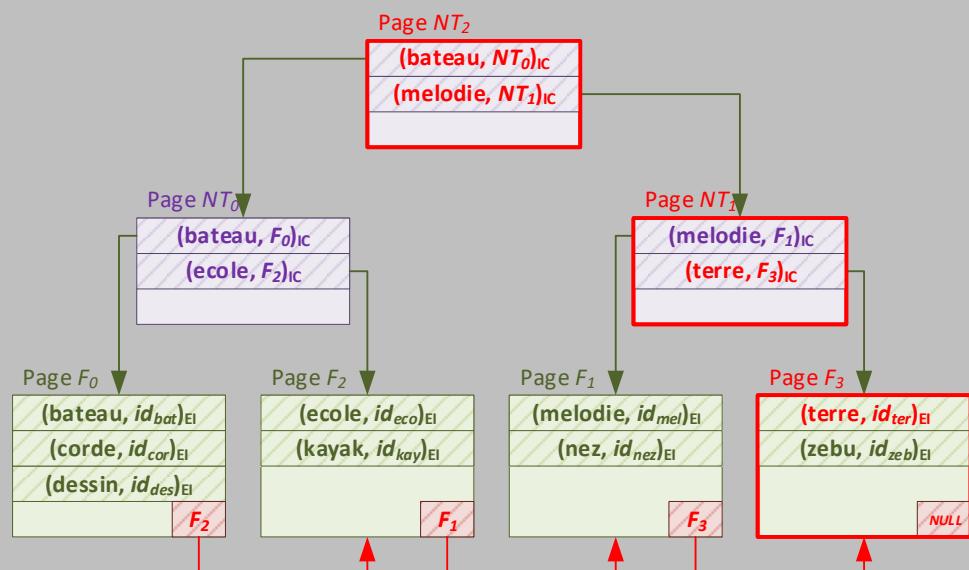


Exemple (suite)

9. On ajoute dans la relation le mot corde, donc l'entrée d'index de valeur corde est insérée dans l'index : il y a de la place dans la page qui contiendrait l'entrée d'index à insérer, i.e. la page F_0 , donc on la rajoute dans cette page (toujours en respectant l'ordonnancement).

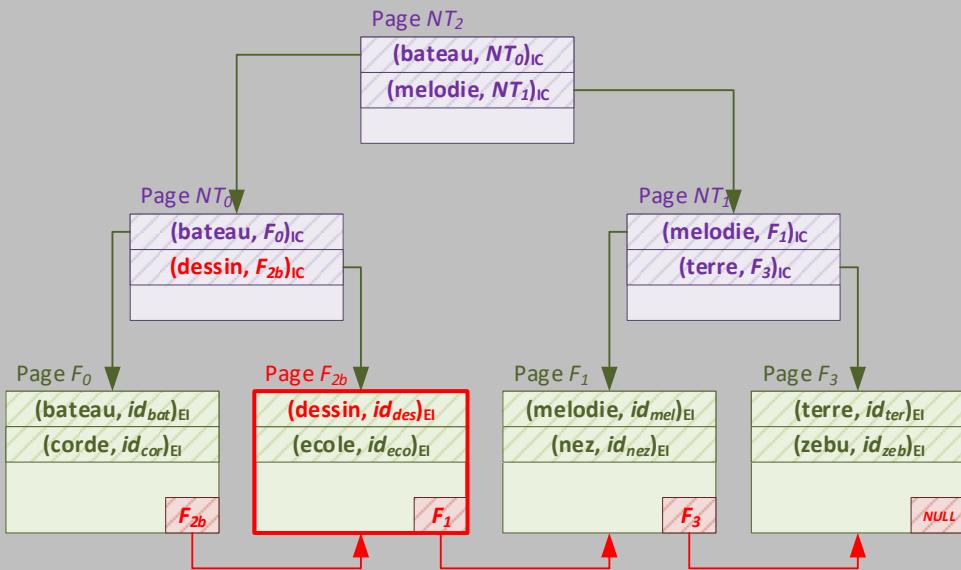


10. On ajoute dans la relation le mot terre, donc l'entrée d'index de valeur terre est insérée dans l'index : si l'on rajoute cette entrée d'index dans la page F_1 où elle devrait être insérée, celle-ci serait trop pleine. Elle est donc divisée en 2 pages, F_1 et F_3 , ce qui fait remonter des informations de circulation au niveau immédiatement supérieur, qui ne contient que la page NT_0 . Ceci pose un nouveau problème : on ne peut pas insérer dans la page NT_0 une nouvelle information de circulation dans l'index. Elle est donc à son tour divisée en 2 pages, NT_0 et NT_1 , ce qui, là encore, fait remonter des informations de circulation au niveau immédiatement supérieur, niveau qui n'existe pas. Il est donc créé (la hauteur de l'arbre augmente de 1), c'est la page NT_2 , et les informations de circulation adéquates peuvent être remontées.

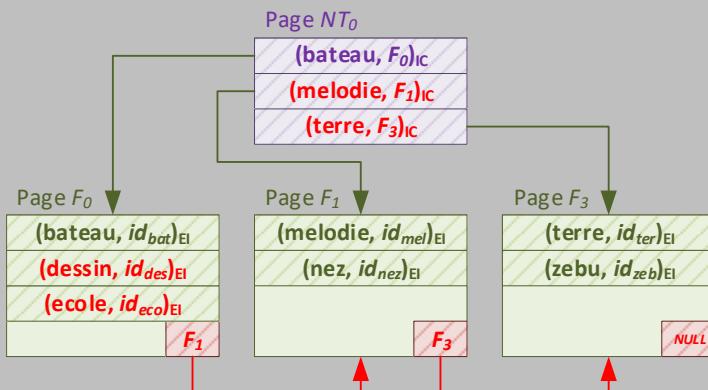


Exemple (fin)

11. On supprime de la relation le mot kayak, donc l'entrée d'index de valeur kayak est supprimée de l'index : l'entrée à supprimer se trouve dans la page F_2 mais cette suppression rendrait la page F_2 trop vide. La page F_2 est donc fusionnée (on la fusionne avec la page F_0 puisqu'elle a le même père que F_2 , contrairement à F_1) mais la page F_1 résultant de la fusion est trop pleine. Celle-ci est alors immédiatement divisée, en F_1 et F_{2b} . Au fur et à mesure de ces fusions/suppressions, les informations de circulation correspondantes sont « adaptées » niveau immédiatement supérieur, c'est-à-dire dans NT_0 (ce qui ne pose ici aucun problème).



12. Enfin, on supprime de la relation le mot corde, dont l'entrée d'index de valeur corde est supprimée de l'index : l'entrée à supprimer se trouve dans la page F_0 mais cette suppression rendrait la page F_0 trop vide. La page F_0 est donc fusionnée (on la fusionne avec la page F_2 puisque c'est sa seule sœur directe). L'information de circulation correspondante est supprimée de NT_0 , qui devient à son tour trop vide et est donc fusionnée avec son unique sœur NT_1 . Là encore, l'information de circulation correspondante est supprimée de NT_2 qui devient elle aussi trop vide et est donc purement et simplement supprimée puisque c'est la racine (la hauteur de l'arbre diminue donc de 1).



8.2.1.4 Performances d'un arbre B+

Nous en avons déjà parlé, l'intérêt d'un index est d'accélérer la recherche de n-uplets en fonction d'une valeur v donnée du constituant indexé.

Remarque

Usuellement, cette recherche se fait grâce à une égalité : on recherche tous les n-uplets tels que leur constituant indexé c vérifie une égalité avec une valeur v ($c = v$). Dans ce cas, pas de souci : on fait la recherche au sein de l'arbre B+ telle qu'on l'a décrite plus haut.

Mais, on recherche aussi parfois tous les n-uplets tels que leur constituant indexé c est borné par 2 valeurs v_{min} et v_{max} ($v_{min} \leq c \leq v_{max}$ ¹⁶⁴). C'est dans ce cas que l'on perçoit le mieux la différence entre un arbre B et un arbre B+ :

- Avec un arbre B : il faut alors très souvent faire plusieurs recherches dans l'arbre (donc plusieurs parcours de cet arbre de haut en bas)...
 1. On cherche l'entrée d'index associée à une valeur v immédiatement supérieure ou égale à v_{min} .
 2. Les entrées d'index étant triées, on « récupère » celle-là et toutes les suivantes au sein de la même page (tant qu'elles sont associées à une valeur v inférieure ou égale à v_{max}). Si on n'a pas « atteint » v_{max} , on note la valeur v_{last} associée à la dernière entrée d'index récupérée dans cette page.
 3. On refait une recherche (depuis la racine, donc) d'une entrée d'index associée à une valeur v immédiatement supérieure strictement à v_{last} . On reprend alors à l'étape 2.
- Avec un arbre B+ : le chaînage des feuilles permet de n'avoir à réaliser qu'une seule recherche dans l'arbre (donc un seul parcours de cet arbre de haut en bas)...
 1. On cherche l'entrée d'index associée à une valeur v immédiatement supérieure ou égale à v_{min} .
 2. Les entrées d'index étant triées et les feuilles étant chaînées entre elles (en conservant l'ordonnancement des entrées d'index qu'elles contiennent), on « récupère » cette entrée d'index-là et toutes les suivantes (tant qu'elles sont associées à une valeur v inférieure ou égale à v_{max}) au sein de la même page et des pages suivantes (en se servant du chaînage entre les feuilles pour passer de l'une à l'autre, donc).

Les arbres B+ montrent donc leur plus grande efficacité sur les arbres B notamment dans les opérations de recherche basées sur une inégalité et ce en ayant mis en place un simple chaînage entre leurs feuilles.

Il est donc entendu que l'usage d'un index permet d'accélérer l'accès aux n-uplets de la relation indexée. Mais, l'usage d'un index a lui-même un coût (l'index étant constitué de pages, celles-ci sont donc à transférer et c'est bien ce qui est coûteux¹⁶⁵). L'idée est donc que le coût d'usage de l'index soit inférieur (si possible très inférieur) au gain qu'il permet lors de l'accès aux n-uplets de la relation indexée...

¹⁶⁴ Que les comparaisons soient strictes ou non ne change pas l'explication donnée ici.

¹⁶⁵ Rappelons que l'on estime négligeable le coût des opérations faites en mémoire de travail.

8.2.1.4.1 Minimiser le coût d'usage d'un index / en arbre B+ : bien fixer $e_{max}(I(R, c))$

Optimiser le coût d'usage d'un index $I(R, c)$ en arbre B+ revient principalement à minimiser le nombre de ses pages. Et, pour cela, la détermination de son ordre $e_{max}(I(R, c))$ est, bien sûr, importante : il est nécessaire de fixer $e_{max}(I(R, c))$ (si sa valeur n'est pas préalablement imposée, bien sûr) de façon que les nœuds puissent contenir le plus possible d'enregistrements d'index. Ainsi, à nombre égal d'enregistrements d'index, si on peut en mettre plus dans chaque nœud, on a besoin de moins de nœuds *a priori*¹⁶⁶ pour les stocker.



Remarque

N'oubliez pas que les enregistrements d'index peuvent être de 2 natures, donc de 2 tailles différentes :

- Des entrées d'index (v, a) où v est une valeur du constituant indexé et a est une liste d'adresses logiques de n-uplets,
- Des informations de circulation dans l'index (v, p) où v est une valeur du constituant indexé et p est un identificateur de page.

Or, c'est bien la même valeur de $e_{max}(I(R, c))$ qui va fixer :

- Le nombre maximal d'entrées d'index que l'on peut mettre dans une feuille,
- Le nombre maximal d'informations de circulation dans l'index que l'on peut mettre dans un nœud non-terminal.

Il faudra donc, pour calculer la « meilleure » valeur de $e_{max}(I(R, c))$ possible, faire ses calculs à partir de la plus grande des tailles d'enregistrements d'index (usuellement, il s'agit de celle des entrées d'index¹⁶⁷).

Dans la suite de cette partie, nous allons considérer que $e_{max}(I(R, c))$ est fixé « au mieux »...

8.2.1.4.2 Considérations générales sur les performances d'un arbre B+

On distingue usuellement, dans les index, les opérations de lecture (recherche) d'un côté et les opérations d'insertion ou de suppression de l'autre. Ainsi, pour les opérations de lecture, le nombre de lectures de pages pour accéder à une entrée d'index est égal au nombre de nœuds de la branche dans laquelle se trouve l'entrée d'index recherchée (*i.e.* à la hauteur de l'arbre + 1), ce qui correspond au nombre de « niveaux » dans l'arbre. De même, le coût des opérations d'insertion et de suppression est proportionnel à la hauteur de l'arbre.

Reste que, dans le cas des arbres B+, on ne sait pas calculer une hauteur moyenne donc un coût moyen ☺ En revanche, on sait calculer ce que cela donne (hauteur et donc coût) dans 2 cas limites : le pire et le meilleur des cas¹⁶⁸.



Remarque

On ne va traiter dans la suite que les opérations de lecture (la base du raisonnement étant sensiblement la même pour les opérations d'insertion et de suppression).

¹⁶⁶ Vous comprendrez ce « *a priori* » dans la suite et, surtout, en TD...

¹⁶⁷ Mais ça ne coûte rien de vérifier que ce soit bien le cas.

¹⁶⁸ La présentation faite ci-après de ces cas ne prend pas en compte la possibilité d'épingler des pages en mémoire cache. Il est cependant fréquent d'épingler en mémoire cache la racine d'un arbre B+.

8.2.1.4.3 Coût d'une lecture dans « le pire des cas »

On est dans le pire des cas quand le coût est maximal. Comme nous venons de le voir, ce coût est lié à la hauteur de l'arbre (plus elle est élevée et plus le coût est élevé). Donc, le pire des cas correspond au cas où l'arbre est le plus haut possible (*i.e.* de hauteur maximale).



Question

Quand la hauteur de l'arbre B+ est-elle maximale ?

Réponse

Cette hauteur est maximale quand il y a un maximum de pages à « placer » dans l'arbre (*i.e.* un maximum de feuilles et un maximum de nœuds non-terminaux aux autres niveaux de l'arbre). L'ordre $e_{max}(l(R, c))$ de l'arbre étant fixé, la hauteur est donc maximale quand les nœuds de l'arbre sont le plus vides possible. L'idée du raisonnement qui suit est de se rapprocher le plus possible de ces nombres *a minima* (si ce n'est de les atteindre), bien sûr « par le haut » (puisque ce sont des minima).

On va raisonner en inférant sur chaque niveau de l'arbre (le niveau 1 étant celui de la racine et le niveau n_{max} étant celui des feuilles). Ainsi, à chaque niveau n , on note :

- Combien de nœuds $Nb_{nœuds}(n)$ se trouvent à ce niveau,
- Le nombre minimal $NbMax_{enr/nmin}(n)$ d'enregistrements d'index que peut contenir chacun des nœuds de ce niveau,
- Le nombre minimal $Nb_{enrTotmin}(n)$ d'enregistrements d'index au total à ce niveau,



Remarque

Fort logiquement, on a :

$$Nb_{enrTotmin}(n) = Nb_{nœuds}(n) \times NbMax_{enr/nmin}(n)$$

Équation 37. Nombre total d'enregistrements d'index au niveau n d'un arbre B+ (pire des cas)

- Le type d'enregistrements d'index $Type_{enr}(n)$ que l'on trouve à ce niveau.



Remarque

On note les 2 types d'enregistrements d'index comme suit :

- « *IC* » pour les informations de circulation dans l'index,
- « *EI* » pour les entrées d'index.

En commençant par la racine, on peut inférer pour tout niveau (ici noté i) et, donc, traiter le niveau des feuilles, le « lien » entre 2 niveaux successifs se faisant comme suit : si on a, au niveau i , n informations de circulation au total à ce niveau, alors on a forcément, au niveau $i + 1$, n nœuds (chaque information de circulation « désignant » un et un seul nœud au niveau immédiatement inférieur).

Niv. n	Nb _{noeuds(n)}	Nb _{Max_{enrlnd/nmin(n)}}	Nb _{enrlndTotmin(n)}	Type _{enr(n)}
1	1	2	2	IC
2	2	$\frac{e_{max}(I(R, c)) + 1}{2}$	$e_{max}(I(R, c)) + 1$	IC
3	$e_{max}(I(R, c)) + 1$	$\frac{e_{max}(I(R, c)) + 1}{2}$	$\frac{(e_{max}(I(R, c)) + 1)^2}{2}$	IC
4	$\frac{(e_{max}(I(R, c)) + 1)^2}{2}$	$\frac{e_{max}(I(R, c)) + 1}{2}$	$\frac{(e_{max}(I(R, c)) + 1)^3}{4}$	IC
5	$\frac{(e_{max}(I(R, c)) + 1)^3}{4}$	$\frac{e_{max}(I(R, c)) + 1}{2}$	$\frac{(e_{max}(I(R, c)) + 1)^4}{8}$	IC
...	IC
i	$2 \times \left(\frac{e_{max}(I(R, c)) + 1}{2} \right)^{i-2}$	$\frac{e_{max}(I(R, c)) + 1}{2}$	$2 \times \left(\frac{e_{max}(I(R, c)) + 1}{2} \right)^{i-1}$	IC
...	IC
n_{max}	$2 \times \left(\frac{e_{max}(I(R, c)) + 1}{2} \right)^{n_{max}-2}$	$\frac{e_{max}(I(R, c)) + 1}{2}$	$2 \times \left(\frac{e_{max}(I(R, c)) + 1}{2} \right)^{n_{max}-1}$	EI

Tableau 38. Raisonnement sur le contenu d'un arbre B+ (pire des cas)

On sait donc que, en théorie et dans le pire des cas, on peut mettre dans l'arbre B+ un nombre total d'entrées d'index dans ses feuilles (niveau n_{max}) égal à :

$$Nb_{EITot} = 2 \times \left(\frac{e_{max}(I(R, c)) + 1}{2} \right)^{n_{max}-1}$$

Équation 38. Nombre d'entrées d'index dans les feuilles d'un arbre B+ (pire des cas)

On souhaite en réalité placer dans cet arbre B+ un nombre d'entrées d'index égal à N . On se demande donc le nombre de niveaux n_{max} nécessaires. Donc, n_{max} est le plus grand entier tel que :

$$\begin{aligned} N &\geq 2 \times \left(\frac{e_{max}(I(R, c)) + 1}{2} \right)^{n_{max}-1} \\ \Rightarrow \frac{N}{2} &\geq \left(\frac{e_{max}(I(R, c)) + 1}{2} \right)^{n_{max}-1} \\ \Rightarrow \log_{\left(\frac{e_{max}(I(R, c)) + 1}{2} \right)} \left(\frac{N}{2} \right) &\geq n_{max} - 1 \\ \Rightarrow \log_{\left(\frac{e_{max}(I(R, c)) + 1}{2} \right)} \left(\frac{N}{2} \right) + 1 &\geq n_{max} \end{aligned}$$

$Puisque \log_n(x) = \frac{\log x}{\log n}$

$$\Rightarrow n_{max} \leq \left(\frac{\log \left(\frac{N}{2} \right)}{\log \left(\frac{e_{max}(I(R, c)) + 1}{2} \right)} \right) + 1 \text{ et } h_{max} \leq \left(\frac{\log \left(\frac{N}{2} \right)}{\log \left(\frac{e_{max}(I(R, c)) + 1}{2} \right)} \right)$$

$Puisque h_{max} = n_{max} - 1$

Équation 39. Niveaux et hauteur d'un arbre B+ pour y stocker N entrées d'index (pire des cas)



Exemple

Si l'on suppose que l'on peut stocker au maximum 99 enregistrements d'index dans chaque nœud de l'arbre ($e_{max}(I(R, c)) = 99$) et qu'il y a 900 000 entrées d'index enregistrées dans l'index ($N = 900 000$), alors, n_{max} est le plus grand entier tel que :

$$\begin{aligned} n_{max} &\leq \left(\frac{\log\left(\frac{900\ 000}{2}\right)}{\log\left(\frac{99+1}{2}\right)} \right) + 1 \\ \Rightarrow n_{max} &\leq \left(\frac{\log(450\ 000)}{\log(50)} \right) + 1 \\ \Rightarrow n_{max} &\leq (3,327) + 1 \\ \Rightarrow n_{max} &\leq 4,427 \\ \Rightarrow n_{max} &= 4 \text{ et } h_{max} = 3 \end{aligned}$$

Ainsi, pour un arbre B+ d'ordre $e_{max}(I(R, c)) = 99$, 4 lectures de page sont nécessaires dans le pire des cas pour retrouver une entrée d'index parmi 900 000 ! 😊



Remarque

Ce calcul n'indique **en aucun cas** le nombre de nœuds effectifs dans l'arbre, juste sa hauteur (et, donc le nombre de niveaux qu'il contient ainsi que ses performances).

Le calcul du nombre de pages occupées par un arbre B+ repose sur un raisonnement similaire mais « pris à l'envers » (i.e. des feuilles vers la racine : on arrivera forcément à un niveau où 1 seul nœud sera nécessaire, c'est donc le niveau racine). Le nombre de pages occupées par l'arbre est la somme du nombre de nœuds à chaque niveau de cet arbre :

Niv. n	Nb_{enrlndTotmin}(n)	Type_{enr}(n)	NbMax_{enrlnd/nmin}(n)	Nb_{noeuds}(n)
n_{max}	N	EI	$\frac{e_{max}(I(R, c)) + 1}{2}$	$\frac{N}{\left(\frac{e_{max}(I(R, c)) + 1}{2}\right)}$ ↗
$n_{max} - 1$	$\frac{N}{\left(\frac{e_{max}(I(R, c)) + 1}{2}\right)}$	IC	$\frac{e_{max}(I(R, c)) + 1}{2}$	$\frac{N}{\left(\frac{e_{max}(I(R, c)) + 1}{2}\right)^2}$ ↗
...	...	IC
$n_{max} - i$	$\frac{N}{\left(\frac{e_{max}(I(R, c)) + 1}{2}\right)^i}$	IC	$\frac{e_{max}(I(R, c)) + 1}{2}$	$\frac{N}{\left(\frac{e_{max}(I(R, c)) + 1}{2}\right)^{i+1}}$ ↗
...	...	IC
1	???	IC	2	1

Tableau 39. Raisonnement sur le nombre de nœuds occupés par un arbre B+ (pire des cas)



Exemple

Sur le même exemple d'un arbre B+ contenant 900 000 entrées d'index et d'ordre $e_{max}(I, (R, c)) = 99$:

Niv. n	$Nb_{enrIndTotmin}(n)$	Type _{enr} (n)	$NbMax_{enrInd/nmin}(n)$	$Nb_{noeuds}(n)$
???	900 000	EI	50	18 000
???	18 000	IC	50	360
???	360	IC	50	7
???	7	IC	50	1

Cet arbre contient donc $1 + 7 + 360 + 18 000 = 18 368$ pages (dont 18 000 pour ses entrées d'index et 368 pour ses informations de circulation) :

Notez que nous retrouvons bien un arbre sur 4 niveaux, donc de hauteur 3 😊

8.2.1.4.4 Coût d'une lecture dans « le meilleur des cas »

On est dans le meilleur des cas quand le coût est minimal. Comme nous l'avons précédemment évoqué, ce coût est lié à la hauteur de l'arbre (plus elle est petite et plus le coût est petit). Donc, le meilleur des cas correspond au cas où l'arbre est le plus petit possible (*i.e.* de hauteur minimale).



Question

Quand la hauteur de l'arbre B+ est-elle minimale ?

Réponse

Cette hauteur est minimale quand il y a un minimum de pages à « placer » dans l'arbre (*i.e.* un minimum de feuilles et un minimum de noeuds non-terminaux aux autres niveaux de l'arbre). L'ordre $e_{max}(I(R, c))$ de l'arbre étant fixé, la hauteur est donc minimale quand les noeuds de l'arbre sont le plus pleins possible. L'idée du raisonnement qui suit est de se rapprocher le plus possible de ces valeurs *a maxima* (si ce n'est de les atteindre), bien sûr « par le bas » (puisque ce sont des maxima).

Comme pour le cas précédent, on va raisonner en inférant sur chaque niveau de l'arbre (le niveau 1 étant celui de la racine et le niveau n_{min} étant celui des feuilles). Ainsi, à chaque niveau n , on note :

- Combien de noeuds $Nb_{noeuds}(n)$ se trouvent à ce niveau,
- Le nombre maximal $NbMax_{enr/nmax}(n)$ d'enregistrements d'index que peut contenir chacun des noeuds de ce niveau,
- Le nombre maximal $Nb_{enrTotmax}(n)$ d'enregistrements d'index au total à ce niveau,



Remarque

Fort logiquement, on a toujours :

$$Nb_{enrTotmax}(n) = Nb_{noeuds}(n) \times NbMax_{enr/nmax}(n)$$

Équation 40. Nombre total d'enregistrements d'index au niveau n d'un arbre B+ (meilleur des cas)

- Le type d'enregistrements d'index $Type_{enr}(n)$ que l'on trouve à ce niveau.



Remarque

On note toujours les 2 types d'enregistrements d'index comme suit :

- « IC » pour les informations de circulation dans l'index,
- « EI » pour les entrées d'index.

En commençant par la racine, on peut inférer pour tout niveau (ici noté i) et, donc, traiter le niveau des feuilles, le « lien » entre 2 niveaux successifs se faisant, là aussi, comme suit : si on a, au niveau i , n informations de circulation au total à ce niveau, alors on a forcément, au niveau $i + 1$, n nœuds.

Niv. n	$Nb_{noeuds}(n)$	$NbMax_{enrlnd/nmin}(n)$	$Nb_{enrlndTotmin}(n)$	$Type_{enr}(n)$
1	1	$e_{max}(I(R, c))$	$e_{max}(I(R, c))$	IC
2	$e_{max}(I(R, c))$	$e_{max}(I(R, c))$	$e_{max}(I(R, c))^2$	IC
3	$e_{max}(I(R, c))^2$	$e_{max}(I(R, c))$	$e_{max}(I(R, c))^3$	IC
4	$e_{max}(I(R, c))^3$	$e_{max}(I(R, c))$	$e_{max}(I(R, c))^4$	IC
5	$e_{max}(I(R, c))^4$	$e_{max}(I(R, c))$	$e_{max}(I(R, c))^5$	IC
...	IC
i	$e_{max}(I(R, c))^{i-1}$	$e_{max}(I(R, c))$	$e_{max}(I(R, c))^i$	IC
...	IC
n_{min}	$e_{max}(I(R, c))^{n_{min}-1}$	$e_{max}(I(R, c))$	$e_{max}(I(R, c))^{n_{min}}$	EI

Tableau 40. Raisonnement sur le contenu d'un arbre B+ (meilleur des cas)

On sait donc que, en théorie et dans le meilleur des cas, on peut mettre dans l'arbre B+ un nombre total d'entrées d'index dans ses feuilles (niveau n_{min}) égal à :

$$Nb_{EITot} = e_{max}(I(R, c))^{n_{min}}$$

Équation 41. Nombre d'entrées d'index dans les feuilles d'un arbre B+ (meilleur des cas)

On souhaite en réalité placer dans cet arbre B+ un nombre d'entrées d'index égal à N . On se demande donc le nombre de niveaux n_{min} nécessaires. Donc, n_{min} est le plus petit entier tel que...

$$N \leq e_{max}(I(R, c))^{n_{min}}$$

$$\Rightarrow \log_{e_{max}(I(R, c))}(N) \leq n_{min}$$

$$\Rightarrow \frac{\log N}{\log e_{max}(I(R, c))} \leq n_{min}$$

$$\Rightarrow n_{min} \geq \frac{\log N}{\log e_{max}(I(R, c))} \text{ et } h_{min} \geq \left(\frac{\log N}{\log e_{max}(I(R, c))} \right) - 1$$

Puisque $\log_n(x) = \frac{\log x}{\log n}$

Puisque $h_{min} = n_{min} - 1$

Équation 42. Niveaux et hauteur d'un arbre B+ pour y stocker N entrées d'index (meilleur des cas)

Exemple

En supposant toujours que $e_{max}(I(R, c)) = 99$ et qu'il y a 900 000 entrées d'index enregistrées dans l'index ($N = 900\ 000$), alors, n_{min} est le plus petit entier tel que :



$$n_{min} \geq \frac{\log 900\ 000}{\log 99}$$

$$\Rightarrow n_{min} \geq 2,983$$

$$\Rightarrow n_{min} = 3 \text{ et } h_{min} = 2$$

Ainsi, pour un arbre B+ d'ordre $e_{max}(I(R, c)) = 99$, 3 lectures de page sont nécessaires dans le meilleur des cas pour retrouver une entrée d'index parmi 900 000 ! 😊



Remarque

Là encore, ce calcul n'indique **en aucun cas** le nombre de noeuds effectifs dans l'arbre, juste sa hauteur (et, donc le nombre de niveaux qu'il contient ainsi que ses performances).

Le calcul du nombre de pages occupées par un arbre B+ repose là aussi sur un raisonnement similaire mais « pris à l'envers » (i.e. des feuilles vers la racine) :

Niv. n	Nb _{enrlndTotmin(n)}	Type _{enr(n)}	NbMax _{enrlnd/nmin(n)}	Nb _{noeuds(n)}
n_{min}	N	EI	$e_{max}(I(R, c))$	$\frac{N}{e_{max}(I(R, c))} \nearrow$
$n_{min} - 1$	$\frac{N}{e_{max}(I(R, c))}$	IC	$e_{max}(I(R, c))$	$\frac{N}{e_{max}(I(R, c))^2} \nearrow$
...	...	IC
$n_{min} - i$	$\frac{N}{e_{max}(I(R, c))^i}$	IC	$e_{max}(I(R, c))$	$\frac{N}{e_{max}(I(R, c))^{i+1}} \nearrow$
...
1	???	IC	$e_{max}(I(R, c))$	1

Tableau 41. Raisonnement sur le nombre de noeuds occupés par un arbre B+ (meilleur des cas)



Exemple

Sur le même exemple (900 000 entrées d'index, ordre $e_{max}(I(R, c)) = 99$) :

Niv. n	Nb _{enrlndTotmin(n)}	Type _{enr(n)}	NbMax _{enrlnd/nmin(n)}	Nb _{noeuds(n)}
???	900 000	EI	99	9 091
???	9 091	IC	99	92
???	92	IC	99	1

Cet arbre contient donc $1 + 92 + 9\ 091 = 9\ 184$ pages (dont 9 091 pour ses entrées d'index et 368 pour ses informations de circulation) ! Notez que nous retrouvons bien un arbre sur 3 niveaux, donc de hauteur 2 😊

8.2.1.5 Paramètres spécifiques à un index $I(R, c)$ en arbre B+

Le tableau ci-dessous résume les paramètres propres à la structuration d'un index en arbre B+ :

- Paramètres liés au contenu des pages de l'arbre B+ :

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$e_{max}(I(R, c))$	-	Ordre de l'arbre B+.
$nbMax_{entrees/page}^{fixe moy min max}(I(R, c))$	entrées d'index par page	<p>Nombre (fixe ou moyen ou minimal ou maximal) d'entrées d'index au plus par page (ce nombre DOIT être borné par $(e_{max}(I(R, c))+1)/2$ et $e_{max}(I(R, c))$ pour les feuilles et par 1 et $e_{max}(I(R, c))$ pour la racine si elle est le seul nœud de l'arbre !) :</p> $nbMax_{entrees/page}^{fixe moy min max}(I(R, c)) = \frac{T_{page}^{fixe}(*) - T_{idPage}^{fixe}(*)}{T_{entree}^{fixe moy min max}(I(R, c))}$
$nb_{pagesEntrees}^{min max}(I(R, c))$	pages	<p>Nombre (minimal ou maximal) de pages nécessaires au stockage des entrées d'index (uniquement) :</p> <ul style="list-style-type: none"> Si l'arbre est de hauteur maximale : $nb_{pagesEntrees}^{max}(I(R, c)) = \frac{nb_{entrees}^{exact}(I(R, c))}{\left(\frac{e_{max}(I(R, c)) + 1}{2}\right)}$ <ul style="list-style-type: none"> Si l'arbre est de hauteur minimale : $nb_{pagesEntrees}^{min}(I(R, c)) = \frac{nb_{entrees}^{exact}(I(R, c))}{e_{max}(I(R, c))}$
$T_{info}^{fixe moy min max}(I(R, c))$	octets	<p>Taille (fixe ou moyenne ou minimale ou maximale) d'une information de circulation dans l'arbre B+ :</p> $T_{info}^{fixe moy min max}(I(R, c)) = T_{valAtt}^{fixe moy min max}(R, c) + T_{idPage}^{fixe}(*)$
$nbMax_{infos/page}^{fixe moy min max}(I(R, c))$	Informations de circulation par page	<p>Nombre (fixe ou moyen ou minimal ou maximal) d'entrées d'informations de circulation au plus par page (ce nombre DOIT être borné par $(e_{max}(I(R, c))+1)/2$ et $e_{max}(I(R, c))$ pour les nœuds non-terminaux et par 2 et $e_{max}(I(R, c))$ pour la racine !) :</p> $Nb_{infos/page}^{fixe moy min max}(I(R, c)) = \frac{T_{page}^{fixe}(*)}{T_{info}^{fixe moy min max}(I(R, c))}$

Tableau 42. Performances d'indexation : paramètres spécifiques au contenu des arbres B+

- Paramètres liés aux performances (coût d'usage) d'un arbre B+ :

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$h_{max}(I(R, c))$	-	Hauteur de l'arbre B+ dans le pire des cas : $h_{max}(I(R, c)) \triangleq \left\lceil \frac{\log \left(\frac{N}{2} \right)}{\log \left(\frac{e_{max}(I(R, c)) + 1}{2} \right)} \right\rceil$
$h_{min}(I(R, c))$	-	Hauteur de l'arbre B+ dans le meilleur des cas : $h_{min} \triangleq \left\lceil \frac{\log N}{\log e_{max}(I(R, c))} \right\rceil - 1$
$Coût_{usage}^{min max}(I(R, c))$	Transferts de pages (en lecture)	Coût d'usage (minimal ou maximal) de l'index en arbre B+ : $Coût_{usage}^{min max}(I(R, c)) = h_{min max}(I(R, c)) + 1$

Tableau 43. Performances d'indexation : paramètres spécifiques aux performances des arbres B+

8.2.2 Index à accès par hachage

On l'a vu (cf. Tableau 36), il existe principalement 2 formes d'index : la famille des index arborescents (dont les arbres B+ font partie) et la famille des index à accès par hachage. Comme le nom de cette dernière l'indique, cette forme d'index est basée sur la technique du hachage, technique largement utilisée à de nombreux niveaux en informatique (et, donc, ici, pour l'indexation).

8.2.2.1 Introduction à la technique du hachage

Le hachage (*hash* en anglais) est une technique très usitée en informatique. Grossièrement, on peut considérer que c'est une mise en œuvre du fameux principe algorithmique « diviser pour régner »¹⁶⁹.



En pratique

Vous avez très certainement déjà rencontré la technique du hachage ! C'est le hachage ou un dérivé du hachage qui se trouve notamment derrière les « tables de hachage » ou encore les « tableaux associatifs » ou encore les « tables clé/valeurs ».

L'idée générale du hachage est la suivante : on doit traiter un très gros ensemble E de données. Plutôt que de les traiter « de front » une par une, on choisit de « réduire » ce gros ensemble de données à un ensemble F beaucoup plus petit. Le lien entre ces ensembles E (le gros) et F (le petit) est fait en partitionnant l'ensemble E de départ et en associant à chaque partition un « représentant » dans F (cf. Figure 119).

¹⁶⁹ Principe sur lequel est, par exemple, basée la recherche dichotomique.

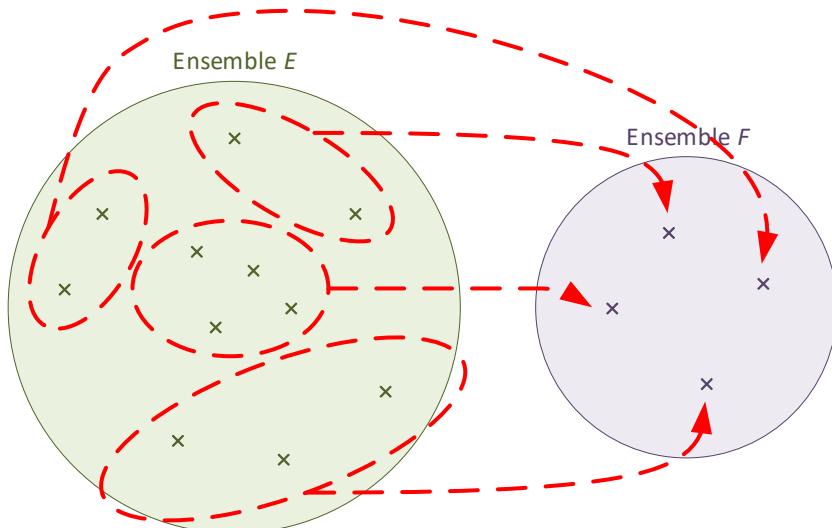


Figure 119. Illustration (grossière) de l'objectif de la technique du hachage

8.2.2.1.1 Concepts basiques pour le hachage

Pour faire le lien entre une partition de l'ensemble E et une donnée de l'ensemble F , on utilise une fonction appelée **fonction de hachage** (*hash function* en anglais) et notée $h()$. Si x est une donnée de l'ensemble E , alors $h(x)$ est une donnée de l'ensemble F et on l'appelle « **code haché** de x » (*hash code* en anglais). Par définition, une fonction de hachage doit assurer que tout élément x de l'ensemble E peut être associé à un élément $h(x)$ de l'ensemble F .



Remarque

L'inverse n'est PAS vrai : un élément de l'ensemble F peut très bien être « rétro-associé » à 0, 1 ou plusieurs éléments de l'ensemble E , peu importe (une fonction de hachage n'est donc pas injective¹⁷⁰ ni surjective¹⁷¹, donc pas bijective non plus¹⁷²). Quand un élément de l'ensemble F est « rétro-associé » à plusieurs éléments de l'ensemble E , on dit qu'on a des **collisions**.



Définition : « fonction de hachage » (*hash function*)

Une **fonction de hachage** est une fonction mathématique $h()$ associant à une donnée d'un ensemble de départ une « signature » dans un ensemble d'arrivée. Plusieurs données de l'ensemble de départ pouvant avoir la même « signature » dans l'ensemble d'arrivée, la fonction de hachage permet de partitionner l'ensemble de départ, chaque partition étant représentée dans l'ensemble d'arrivée par une signature : celle de tous les éléments de la partition.



Définition : « code haché » (*hash code*)

Le **code haché** d'un élément d'un ensemble de départ est sa « signature » dans l'ensemble d'arrivée. Cette « signature » pouvant être commune à plusieurs éléments de l'ensemble de départ, tous ces éléments-là ont donc le même code haché (et on a des collisions sur ce code haché). On parle aussi parfois de « somme de contrôle » (*checksum* en anglais).

¹⁷⁰ Une fonction $f(x)$ est injective quand $(v_1 \neq v_2) \Rightarrow (f(v_1) \neq f(v_2))$.

¹⁷¹ Une fonction $f(x)$ est surjective quand tous les éléments de l'ensemble d'arrivée ont au moins un antécédent.

¹⁷² Une fonction $f(x)$ est bijective quand elle est simultanément injective et surjective.



Définition : « collision »

Il y a une **collision** quand 2 (ou plus) éléments de l'ensemble de départ ont le même code haché, *i.e.* la même « signature » dans l'ensemble d'arrivée.

Ainsi, au lieu de chercher une donnée x dans l'ensemble E pour la traiter, on cherche dans l'ensemble F le représentant $h(x)$ de la partition de E à laquelle elle appartient et on restreint la recherche de la donnée x à cette partition de l'ensemble E , ce qui est donc logiquement bien plus rapide.



En pratique

Il existe beaucoup de fonctions de hachage. Nous allons nous limiter ici aux fonctions de hachage retournant, quel que soit x l'entier naturel¹⁷³ donné en entrée, un entier naturel $h(x)$ toujours compris entre 0, borne incluse, et un entier naturel N , borne exclue, que l'on aura préalablement, et si possible « astucieusement », choisi : $0 \leq h(x) < N$. Même en se limitant à de telles fonctions de hachage, on en a encore beaucoup à disposition. Mais vous connaissez tous depuis TRÈS longtemps une fonction mathématique basique qui, quel que soit l'entier naturel x donné en entré, retourne un entier naturel $h(x)$ toujours compris entre 0, borne incluse, et un entier naturel N fixé, borne exclue... Il s'agit de... De... De la fonction *modulo*¹⁷⁴ : $x \bmod N$! 😊

8.2.2.1.2 Un hachage « performant », c'est quoi ?

Un hachage est performant quand il optimise bien l'accès à une donnée de l'ensemble de départ E ... C'est entendu, mais encore ? Quand cet accès est-il bien optimisé ? Bien que basique, la présentation du hachage faite ci-dessus donne déjà une bonne idée de ce que peut être un hachage performant :

- Il doit uniformiser le plus possible la taille des partitions faites dans l'ensemble de départ E ,



En pratique

Voilà ce qu'on veut éviter au maximum ici :

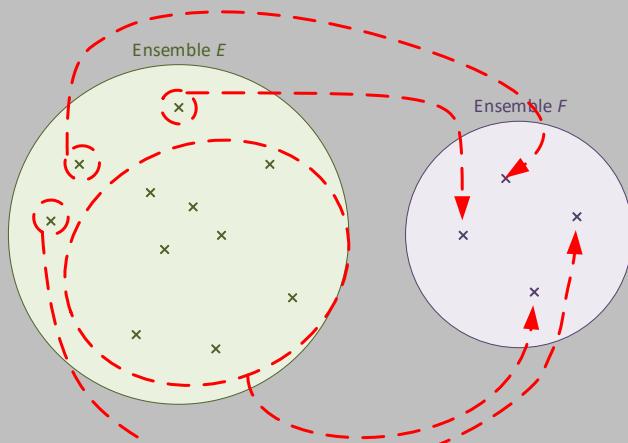


Figure 120. Cas limite de « mauvais hachage » (tailles des partitions très inégales)

¹⁷³ Rappelons que x est un entier naturel si c'est un entier positif ou nul ($x \in \mathbb{N}$).

¹⁷⁴ Rappelons que le modulo $x \bmod N$ est le reste de la division entière de x par N , reste qui, par définition, est toujours compris entre 0, borne incluse, et N , borne exclue.

- Il doit éviter le plus possible les codes hachés « inutilisés » dans l'ensemble d'arrivée F .



En pratique

Et voilà ce qu'on veut éviter au maximum ici :

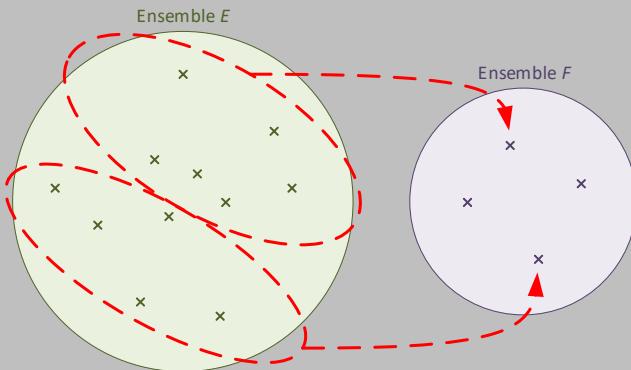


Figure 121. Cas limite de « mauvais hachage » (code hachés inutilisés)

Ainsi, un hachage est considéré comme optimal quand il uniformise la taille des partitions de l'ensemble E (on dit plutôt qu'il « uniformise les collisions » ou qu'il « répartit équiprobablement les collisions ») **ET** quand il répartit les éléments de l'ensemble E dans un nombre de partitions proche du nombre d'éléments de F . Donc, en théorie, si $\text{card}(E)$ est le nombre d'éléments contenus dans E et $\text{card}(F)$ est le nombre d'éléments contenus dans F , le hachage est réellement optimal quand :

- Le nombre de partitions définies par le hachage dans E est égal à $\text{card}(F)$,
- La taille de chacune de ces partitions est proche de $\text{card}(E)/\text{card}(F)$.



En pratique

On n'atteint bien sûr jamais cet optimal théorique, mais on essaie de s'en rapprocher !

8.2.2.1.3 Techniques de hachage basiques

Nous allons présenter ici 2 techniques de hachage qui, quel que soit l'entier naturel x donné en entrée, très basiques mais aussi très « parlantes » :

- La division,
- Le pliage.

Ces 2 techniques usuelles de hachage offrent (globalement sur des volumes importants de données) de bonnes performances : elles répartissent plutôt uniformément les collisions sur l'ensemble des codes hachés et elles évitent plutôt bien les codes hachés inutilisés, le tout restant très simples (donc très rapides) à calculer (pour un ordinateur) 😊

8.2.2.1.3.1 Technique de hachage par division

Il s'agit ici non pas de faire une division mais, plus précisément, de calculer le reste d'une division entière. On fait donc en réalité un *modulo* (nous en avons parlé un peu plus haut).

Question

Mais les données de départ ne sont pas toujours des entiers naturels ! Comment, dès lors, calculer leur *modulo* ?



Réponse

Il est toujours possible de passer d'une donnée de n'importe quel type à un entier naturel, toujours ! Surtout en informatique ! Un moyen simple (mais pas toujours le plus judicieux) est de partir du codage binaire de la donnée et de l'interpréter non pas selon son vrai type mais comme si c'était le codage d'un entier naturel...

Par exemple, en ASCII étendu, les caractères sont codés sur 8 bits (1 octet, on code donc 256 caractères dans cette table). Le codage binaire associé à la lettre « *a* » dans cette table de caractères est $(0110\ 0001)_2$. Ce codage binaire, au lieu d'être interprété comme celui d'un caractère ASCII étendu, peut fort bien être interprété comme celui d'un entier naturel ! Dans le cas présent, il s'agit alors du codage binaire de l'entier naturel 97...

La seule vraie difficulté, ici, est de bien choisir la borne (exclue) N : trop petite ou trop grande et cela réduit, parfois drastiquement, les performances du hachage. 😞



En pratique

Sans rentrer dans des considérations mathématiques guidant un bon choix pour N , on peut dire qu'il est statistiquement démontré qu'il vaut mieux choisir pour valeur de N un nombre premier.

8.2.2.1.3.2 Technique de hachage par pliage (sur b bits)

La technique du pliage est légèrement plus élaborée puisqu'elle travaille sur le codage binaire des données « à hacher » (c'est donc moins naturel à réaliser pour les humains que nous sommes mais c'est, au contraire, encore plus simple à réaliser pour un ordinateur, puisqu'on ne va réaliser ici que des opérations binaires basiques) ! Le principe est le suivant (on souhaite calculer le code haché $h(x)$) :

1. On choisit arbitrairement b un nombre de bits.



Exemple (début)

On choisit par exemple $b = 8$ (pour travailler avec des blocs de 8 bits, soit 1 octet).

2. On découpe le codage binaire de x en « tranches » de b bits, en complétant si nécessaire ses bits de poids forts par des 0 non-significatifs afin d'obtenir des tranches complètes.



Exemple (suite)

Prenons par exemple le codage binaire suivant pour x :

$$x = (100 \ 01101100 \ 11000010 \ 00011010)_2$$

On complète l'écriture du codage binaire de x avec des 0 non-significatifs en bits de poids fort (à gauche) pour avoir un codage sur un multiple de 8 bits :

$$x = (00000100 \ 01101100 \ 11000010 \ 00011010)_2$$

On découpe cette écriture en tranches de 8 bits :

Tranche PF	Tranche Pf
Tranche 1	Tranche 2	Tranche 3	Tranche 4
00000100	01101100	11000010	00011010

3. On empile ces tranches de b bits les unes sous les autres (celle de poids fort, PF , en haut jusqu'à celle de poids faible, Pf , en bas). Comme on a fait des tranches de b bits, on a donc b colonnes de bits alignés les uns sous les autres.



Exemple (suite)

On empile les tranches de 8 bits les unes sous les autres :

Tranche PF	Tranche 1	0	0	0	0	0	1	0	0
...	Tranche 2	0	1	1	0	1	1	0	0
...	Tranche 3	1	1	0	0	0	0	1	0
Tranche Pf	Tranche 4	0	0	0	1	1	0	1	0

4. Le codage binaire du code haché est codé lui aussi sur b bits et est calculé comme suit : le bit de rang i ($0 \leq i < b$) du codage binaire de $h(x)$ est égal au résultat du ou exclusif (XOR) opéré sur tous les bits de la colonne de rang i obtenue.



Rappels (début)

Voici la table de vérité du ou exclusif (XOR) :

$a XOR b$		b	
		0	1
a	0	0	1
	1	1	0

De plus, où que soient les bits valant 1 dans les suites d'opérations XOR ci-dessous, quel que soit le nombre de bits à 0 dans ces suites d'opérations, et en n'oubliant pas que XOR est associatif et commutatif, on note que :

- $0 \text{ XOR } 0 \text{ XOR } 0 \text{ XOR } 0 \text{ XOR } 0 \dots \text{ XOR } 0 = 0$ (avec aucun bit à 1),
- $1 \text{ XOR } 0 \text{ XOR } 0 \text{ XOR } 0 \text{ XOR } 0 \dots \text{ XOR } 0 = 1$ (avec 1 bit à 1),
- $1 \text{ XOR } 1 \text{ XOR } 0 \text{ XOR } 0 \text{ XOR } 0 \text{ XOR } 0 \dots \text{ XOR } 0 = 0$ (avec 2 bits à 1),
- $1 \text{ XOR } 1 \text{ XOR } 1 \text{ XOR } 0 \text{ XOR } 0 \text{ XOR } 0 \dots \text{ XOR } 0 = 1$ (avec 3 bits à 1),
- $1 \text{ XOR } 1 \text{ XOR } 1 \text{ XOR } 1 \text{ XOR } 0 \text{ XOR } 0 \dots \text{ XOR } 0 = 0$ (avec 4 bits à 1),
- ...



Rappels (fin)

On peut donc généraliser comme suit :

- Le résultat d'une suite de XOR ne dépend pas du nombre de bits à 0,
- Le résultat d'une suite de XOR dépend de la parité du nombre de bits à 1 :
 - Si ce nombre est pair, le résultat de la suite de XOR vaut 0,
 - Si ce nombre est impair, le résultat de la suite de XOR vaut 1.



Exemple (suite)

On calcule alors le codage binaire de $h(x)$:

Tranche 1	0	0	0	0	0	1	0	0
Tranche 2	0	1	1	0	1	1	0	0
Tranche 3	1	1	0	0	0	0	1	0
Tranche 4	0	0	0	1	1	0	1	0
Code binaire de $h(x)$	1 0 1 1 0 0 0 0							

5. Donc, en faisant le ou exclusif des bits contenus dans une colonne, et ce pour chacune des b colonnes de bits, on obtient bien les b bits du codage binaire de $h(x)$. Donc, on a en fait $h(x)$: il suffit de convertir ce codage binaire en entier naturel !



Exemple (fin)

Le codage binaire du code haché $h(x)$ est donc $(10110000)_2$. Interprété comme un entier naturel, on a donc $h(x) = (176)_{10}$.



Remarque

Le codage binaire de $h(x)$ est écrit sur b bits et est interprété comme le codage d'un entier naturel. Or, par définition, un entier naturel n codé sur b bits est toujours compris entre 0, borne incluse, et 2^b , borne exclue : $0 \leq n < 2^b$. Le pliage est donc bien une fonction de hachage retournant un entier naturel compris entre 0, borne incluse, et un entier naturel N , borne exclue (mais ici on ne choisit pas directement un entier naturel N mais une valeur de b , puis on a $N : N = 2^b$). Ainsi, dans l'exemple précédent, on a choisi de faire des tranches de 8 bits ($b = 8$). Donc, le code haché $h(x)$ est compris dans l'intervalle $0 \leq h(x) < 256$ (puisque $2^8 = 256$).

8.2.2.2 Index mono-critère à accès par hachage statique (IMCAHS)

Un index mono-critère à accès par hachage statique (IMCAHS) peut être organisé de deux façons : avec répertoire (IMCAHSaR) ou sans répertoire (IMCAHSsR). Nous présenterons en détail l'organisation avec répertoire puis, sur un exemple équivalent, l'organisation sans répertoire.

8.2.2.2.1 Organisation AVEC répertoire (IMCAHSaR)

La structure d'index mono-critère à accès par hachage statique avec répertoire (IMCAHSaR) est très utilisée : elle allie en effet simplicité de gestion et performances.

8.2.2.2.1.1 Organisation

Un index mono-critère à accès par hachage statique avec répertoire est défini par :

- Une **fonction de hachage $h()$** ayant une **borne maximale**, un entier naturel, $N : 0 \leq h(x) < N$,
- Un **répertoire de hachage statique R** contenant N cases, indicées de 0 à $N - 1$, chaque case pouvant contenir un identificateur de page (et, donc, « pointer » vers une page),
- Un **ensemble de pages, P** , « pointées » depuis le répertoire de hachage statique R et contenant les entrées d'index.



Définition : « index mono-critère à accès par hachage statique avec répertoire »

Un **index mono-critère à accès par hachage statique avec répertoire** est une structure d'index. Celle-ci prend la forme d'un répertoire de hachage statique R , constituant à lui seul les informations de circulation dans l'index, et d'en ensemble de pages P (organisées en plusieurs listes chaînées de pages : au plus 1 liste chaînée par case du répertoire de hachage statique R) comprenant les entrées d'index (qui sont non ordonnées). Le lien entre le répertoire de hachage statique R et cet ensemble de pages P est fait par le choix d'une fonction de hachage $h()$ retournant un entier naturel tel que $0 \leq h(x) < N$, ce maximum N indiquant aussi le nombre de cases du répertoire de hachage statique R , indicées de 0 à $N - 1$. Toutes les entrées d'index (v, a) ayant le même code haché $h(v) = i$ se trouvent dans une liste chaînée de pages initialement « pointée » par la case d'indice i du répertoire de hachage statique R .



Remarque

Notez que, contrairement à ce qui est fait dans un arbre B+, les entrées d'index ne sont ici PAS triées en fonction de la valeur du constituant indexé à laquelle elles sont associées. Cela serait inutile voire contreproductif au niveau des performances de cette structure d'index.

Exemple

L'exemple ci-dessous donne une idée de cette structure d'index.

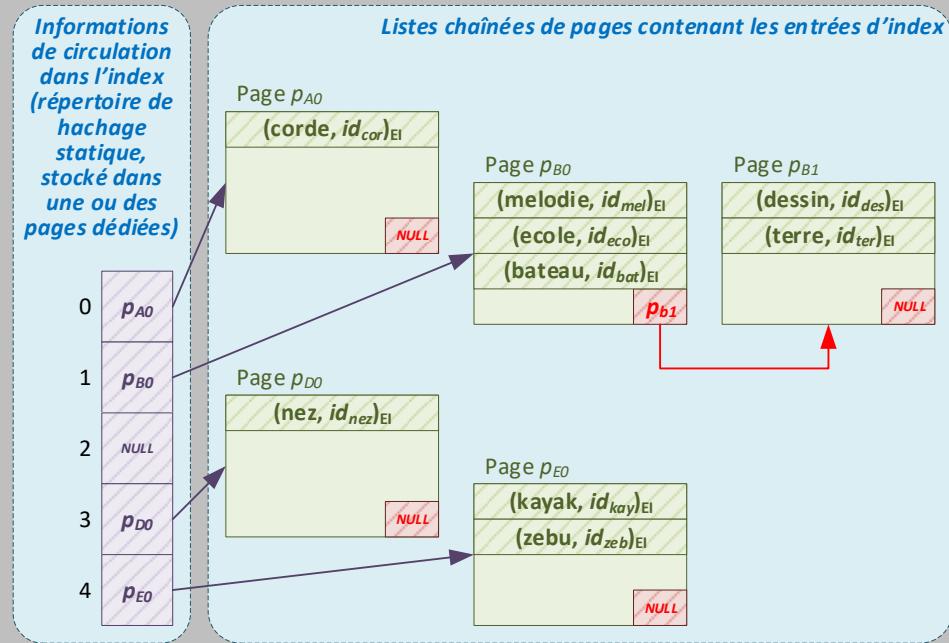


Figure 122. Exemple d'index mono-critère à accès par hachage statique avec répertoire



En interprétant les indices i des cases du répertoire de hachage statique comme des codes hachés $h(x) = i$, on voit bien la répartition des entrées d'index dans les listes chaînées de pages « pointées » depuis les cases du répertoire de hachage statique.

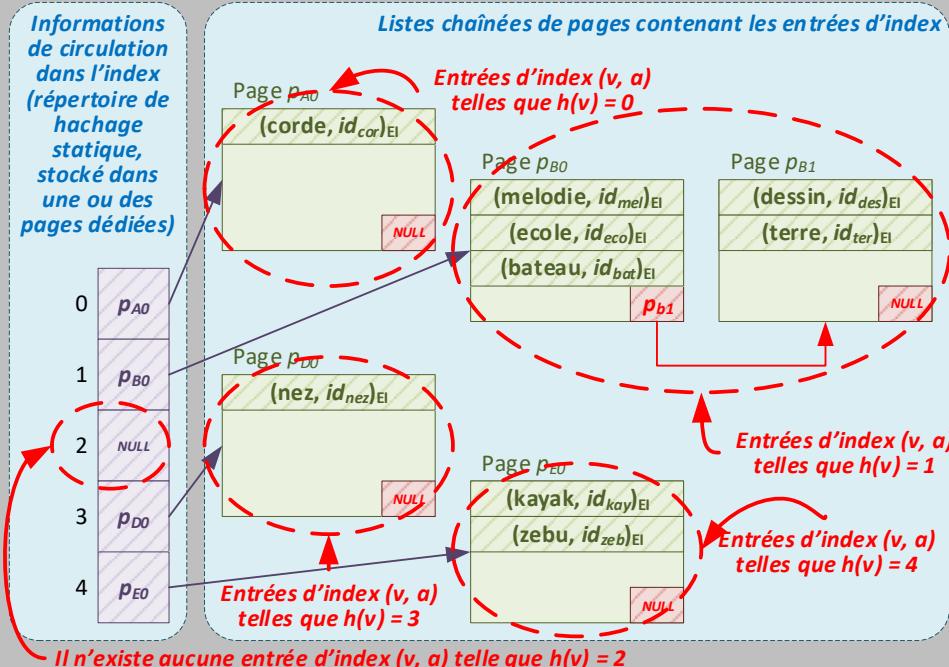


Figure 123. Répartition des entrées d'index (v, a) en fonction de leur code haché $h(v)$

On peut donc globalement voir comme suit les différentes structures de pages intervenant dans le stockage d'un IMCAHS avec répertoire et dans celui de la relation indexée par cet IMCAHSaR :

- Si la relation indexée est en mode d'adressage direct :

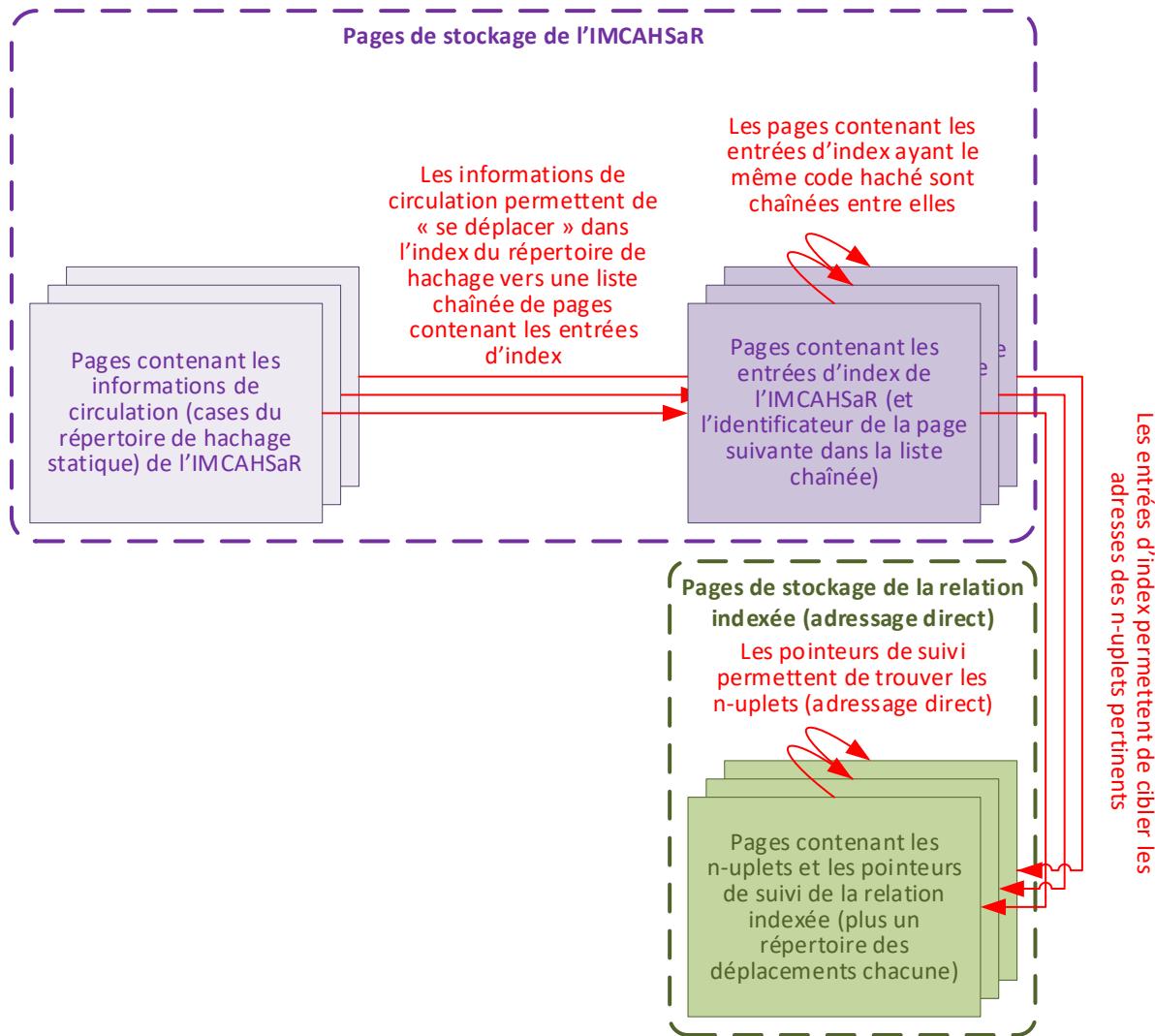


Figure 124. Pages de stockage d'un IMCAHSaR et de la relation indexée (adressage direct)

- Si la relation indexée est en mode d'adressage indirect :

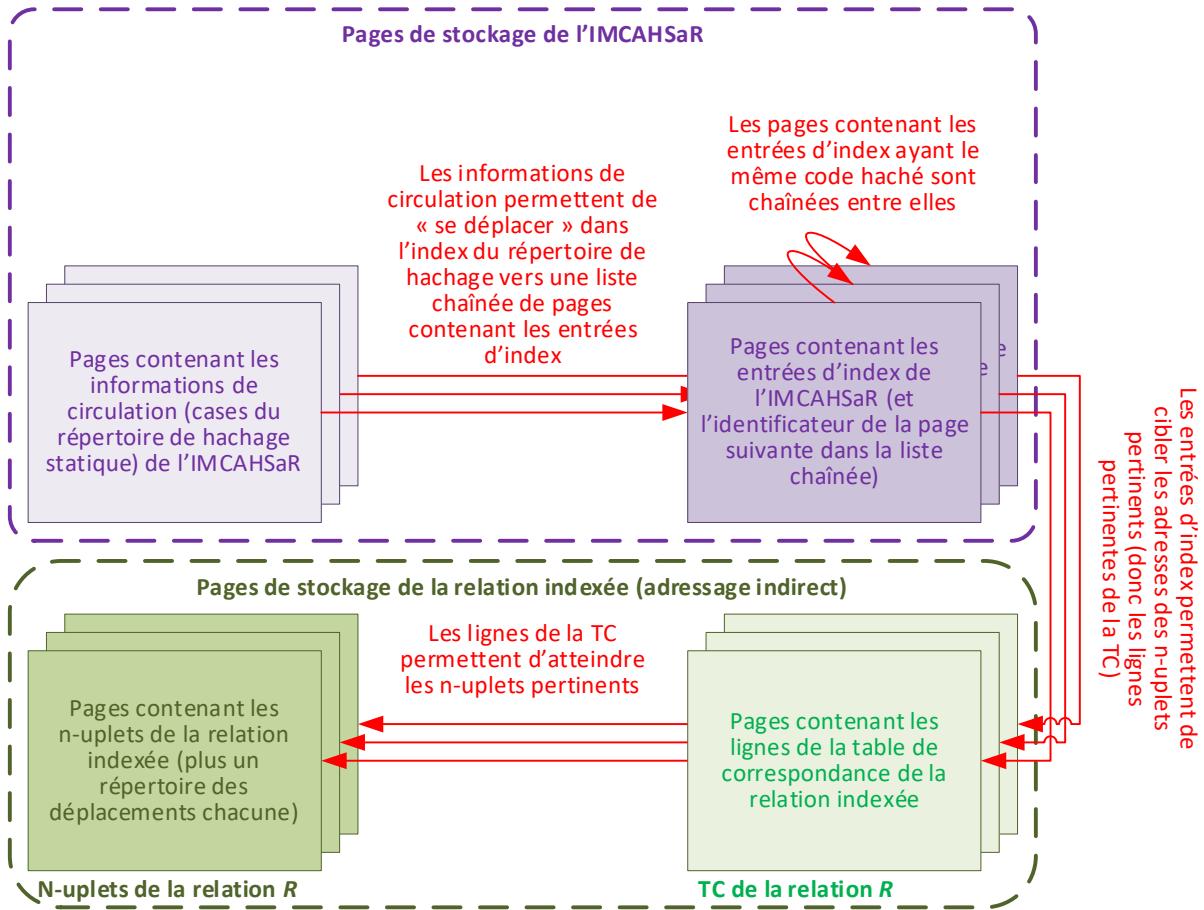


Figure 125. Pages de stockage d'un IMCAHSaR et de la relation indexée (adressage indirect)

8.2.2.2.1.2 Évolution d'un IMCAHSaR : opérations de manipulation

On l'a déjà dit, un index (donc un IMCAHSaR) vit avec la relation qu'il indexe (cf. §8.1.4) :

- *Quand cette relation ne contient aucun n-plet* : l'index est vide (il contient juste le répertoire de hachage statique R , dont toutes les cases sont vides, et l'ensemble de pages P ne contient aucune page du tout),
- *Quand la relation contient des n-plets* : au moins une des cases du répertoire de hachage statique R « pointe » vers une page et l'ensemble de pages P contenant les entrées d'index contient au moins une page.

8.2.2.2.1.2.1 Recherche d'une entrée d'index

La recherche d'une entrée d'index (v, a) se fait d'après la valeur v : on calcule $h(v)$, la valeur obtenue étant l'indice d'une case du répertoire de hachage statique R . On parcourt alors la liste chaînée de pages associée à cette case, séquentiellement, afin d'y trouver l'entrée d'index recherchée. Si on arrive au bout de cette liste chaînée de pages sans l'avoir trouvée, c'est que l'entrée d'index cherchée n'existe pas et, donc, qu'il n'existe pas dans la relation de n-uplet possédant cette valeur v pour le constituant indexé.

En pratique

Le pseudo-algorithme suivant reflète le fonctionnement de la recherche.

Pseudo-algorithme RechercheEIIMCAHSaR

Données d'entrée

v : une valeur

Données de sortie

EI : l'entrée d'index (v, a) recherchée

p : l'identificateur de la page contenant EI

Données intermédiaires

i : l'indice d'une case du répertoire de l'IMCAHSaR

$h()$: la fonction de hachage associée à l'IMCAHSaR

p' : un identificateur de page

$trouve$: un drapeau

Début

$trouve \leftarrow FAUX$

$i \leftarrow h(v)$

$p \leftarrow$ identificateur de page contenu dans la case i

$EI \leftarrow$ 1^{ère} entrée d'index dans p

$p' \leftarrow$ identificateur de page suivante à la fin de p

Tant que ($trouve = FAUX$) **et** ($p' \neq NULL$) **faire**

Si EI est associée à v **alors**

$trouve \leftarrow VRAI$

Sinon

Si p contient d'autres entrées d'index **alors**

$EI \leftarrow$ entrée d'index suivante dans p

Sinon

Si ($p' \neq NULL$) **alors**

$p \leftarrow p'$

$EI \leftarrow$ 1^{ère} entrée d'index dans p

$p' \leftarrow$ id. de page suivante à la fin de p

FinSi

FinSi

FinSi

FinTQ

Fin



8.2.2.2.1.2.2 Insertion d'une entrée d'index

L'insertion d'une entrée d'index (v, a) se déroule globalement comme suit : si cette entrée n'existe pas déjà, on calcule $h(v)$, la valeur obtenue étant l'indice d'une case du répertoire de hachage statique R . On parcourt alors la liste chaînée de pages associée à cette case, séquentiellement, afin d'y trouver la première place libre et d'y placer l'entrée d'index à insérer. S'il n'y a aucune place libre dans cette liste chaînée de pages, on rajoute une page à la fin de cette liste chaînée et on y écrit l'entrée d'index à insérer.

En pratique

Le pseudo-algorithme suivant illustre l'insertion d'une entrée d'index.

Pseudo-algorithme InsertionEIIMCAHSaR

Données d'entrée

EI_{ins} : l'entrée d'index (v, a) à insérer

Données de sortie

p : page dans laquelle l'insertion est réalisée

Données intermédiaires

EI_{ch} : une entrée d'index

i : l'indice d'une case du répertoire de l'IMCAHSaR

$h()$: la fonction de hachage associée à l'IMCAHSaR

Début

Rechercher l'entrée d'index de valeur v^{175}

$EI_{ch} \leftarrow$ entrée d'index trouvée

Si EI_{ch} existe **alors**

Rajouter a aux adresses logiques contenues dans EI_{ch}

Sinon

$i \leftarrow h(v)$

Parcourir la chaîne de pages liées à la case i
jusqu'à en trouver une qui possède une place suffisante pour la nouvelle entrée d'index EI_{ins}

S'il en existe une **alors**

$p \leftarrow$ page dans laquelle l'insertion peut se faire

Y insérer EI_{ins}

Sinon (le bout de la chaîne a été atteint)

Créer une nouvelle page au bout de la chaîne

$p \leftarrow$ nouvelle page ainsi créée

Y insérer EI_{ins}

FinSi

FinSi

Fin



¹⁷⁵ Voir le pseudo-algorithme RechercheEIIMCAHSaR plus haut (cf. §8.2.2.1.2.1).

8.2.2.2.1.2.3 Suppression d'une entrée d'index

Grossièrement, la suppression d'une entrée d'index (v, a) est réalisée comme suit : on recherche l'entrée d'index à supprimer et, effectivement, on la supprime. Si la page qu'elle contient est maintenant vide, on relie directement ce qui la « pointe » (la page précédente ou la case du répertoire qui la « pointait ») à la page qu'elle « pointe » elle-même (éventuellement aucune).

En pratique

Le pseudo-algorithme suivant illustre la suppression d'une entrée d'index.

```

Pseudo-algorithme SuppressionEIMCAHSaR
Données d'entrée
    EI : l'entrée d'index ( $v, a$ ) à supprimer
Données de sortie
    p : la page de laquelle EI est supprimée
Données intermédiaires
    prec : le prédécesseur (page ou case) de p
    succ : le successeur (page) de p
Début
    Rechercher l'entrée d'index de valeur  $v^{176}$ 
    Si elle n'existe pas alors
        p ← NULL
    Sinon
        EI ← entrée d'index ( $v, a$ ) trouvée
        p ← page finale de la recherche (contenant EI)
        Supprimer EI de p
        Si p est maintenant vide alors
            prec ← page ou case « pointant » vers p
            succ ← page « pointée » par p
            Écrire succ en lieu et place de p dans prec
        FinSi
    FinSi
Fin

```



8.2.2.2.1.2.4 Exemple de manipulation d'un IMCAHSaR

On se propose de montrer l'évolution d'un index mono-critère à accès par hachage statique avec répertoire au fur et à mesure de l'ajout/suppression de n-uplets à/de la relation indexée. On suppose que la relation indexée est la relation Dictionnaire (cf. Tableau 37), que cette relation est initialement vide et que l'index que l'on considère est un index primaire ayant la structure d'un index mono-critère à accès par hachage statique avec répertoire. On suppose que, dans cet index, **les pages de l'ensemble de pages P peuvent contenir au maximum 3 entrées d'index** (plus l'identificateur de leur sœur droite, bien sûr). On suppose également que la fonction de hachage utilisée pour définir cet index est $h(v) = (\text{valeur calculée à partir des codes ASCII étendus des caractères de } v) \text{ modulo } 5$.



Remarque

Donc, selon $h(v)$, le répertoire R contient 5 cases indiquées de 0 à 4.

¹⁷⁶ Voir le pseudo-algorithme RechercheEIMCAHSaR plus haut (cf. §8.2.2.2.1.2.1).

Exemple (début)

On commence par donner ci-dessous le code haché des mots qui vont être ajoutés au dictionnaire (dans l'ordre dans lequel ces mots vont être ajoutés) :

Mot (valeur v)	Code haché ($h(v)$)
melodie	1
ecole	1
nez	3
bateau	1
kayak	4
zebu	4
dessin	1
corde	0
terre	1

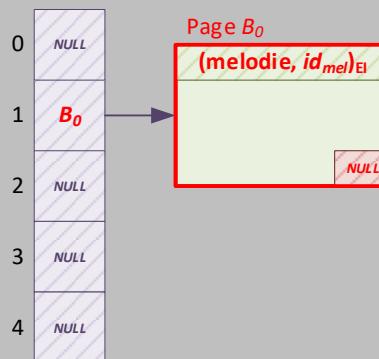
Tableau 44. Codes hachés utilisés pour l'exemple d'IMCAHSaR

La relation va évoluer opération après opération, l'index associé également :

1. Initialement, la relation Dictionnaire est vide, l'index aussi : il ne contient alors que le répertoire de hachage statique de 5 cases, indicées de 0 à 4, ces cases ne « pointent » aucune page.

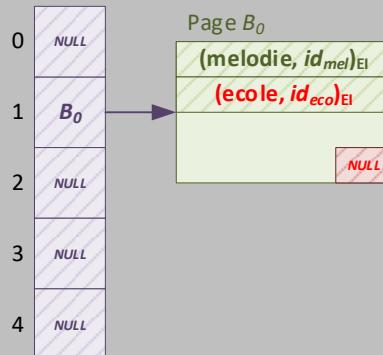


2. On ajoute dans la relation le mot `melodie`, donc l'entrée d'index associée à la valeur `melodie` est insérée dans l'index : son code haché $h(melodie)$ vaut 1 ; la case d'indice 1 du répertoire de hachage statique ne « pointant » vers aucune page, la page B_0 est créée, « pointée » depuis la case d'indice 1 et l'entrée d'index y est rajoutée.

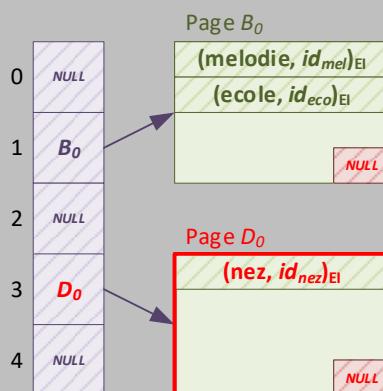


Exemple (suite)

3. On ajoute dans la relation le mot `ecole`, donc l'entrée d'index associée à la valeur `ecole` est insérée dans l'index : son code haché $h(\text{ecole})$ vaut 1 et il y a de la place dans la page (unique pour le moment) « pointée » par la case d'indice 1 du répertoire de hachage statique. On ajoute donc l'entrée d'index dans la première place libre de cette page.

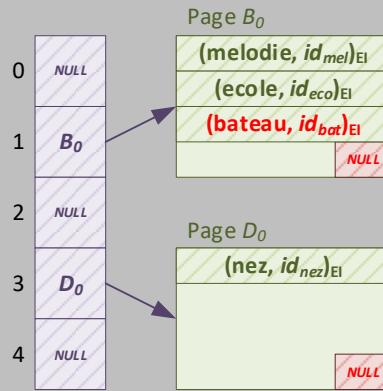


4. On ajoute dans la relation le mot `nez`, donc l'entrée d'index associée à la valeur `nez` est insérée dans l'index : son code haché $h(\text{nez})$ vaut 3 ; la case d'indice 3 du répertoire de hachage statique ne « pointant » vers aucune page, la page D_0 est créée, « pointée » depuis la case d'indice 3 et l'entrée d'index y est rajoutée.

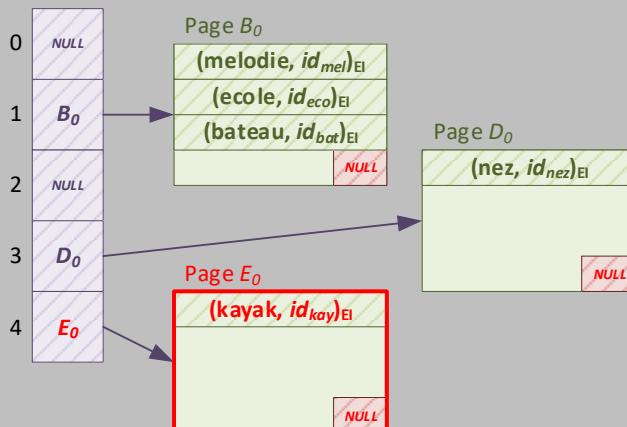


Exemple (suite)

5. On ajoute dans la relation le mot bateau, donc l'entrée d'index de valeur bateau est insérée dans l'index : son code haché $h(\text{bateau})$ vaut 1 et il y a de la place dans la page (unique pour le moment) « pointée » par la case d'indice 1 du répertoire de hachage statique. On ajoute donc l'entrée d'index dans la première place libre de cette page.

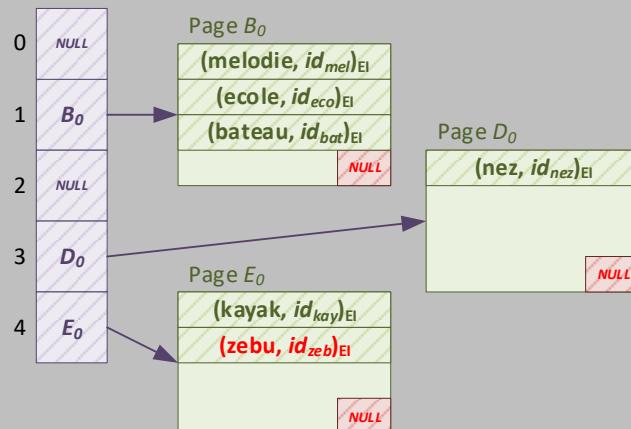


6. On ajoute dans la relation le mot kayak, donc l'entrée d'index de valeur kayak est insérée dans l'index : son code haché $h(\text{kayak})$ vaut 4 ; la case d'indice 4 du répertoire de hachage statique ne « pointant » vers aucune page, la page E_0 est créée, « pointée » depuis la case d'indice 4 et l'entrée d'index y est rajoutée.

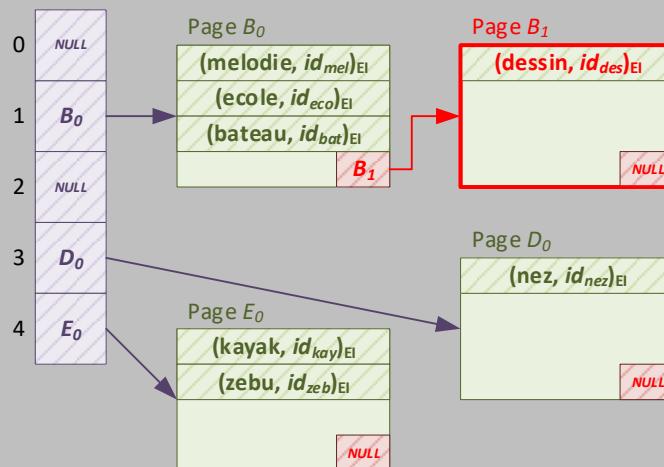


Exemple (suite)

7. On ajoute dans la relation le mot zebu, donc l'entrée d'index de valeur zebu est insérée dans l'index : son code haché $h(\text{zebu})$ vaut 4 et il y a de la place dans la page (unique pour le moment) « pointée » par la case d'indice 4 du répertoire de hachage statique. On ajoute donc l'entrée d'index dans la première place libre de cette page.

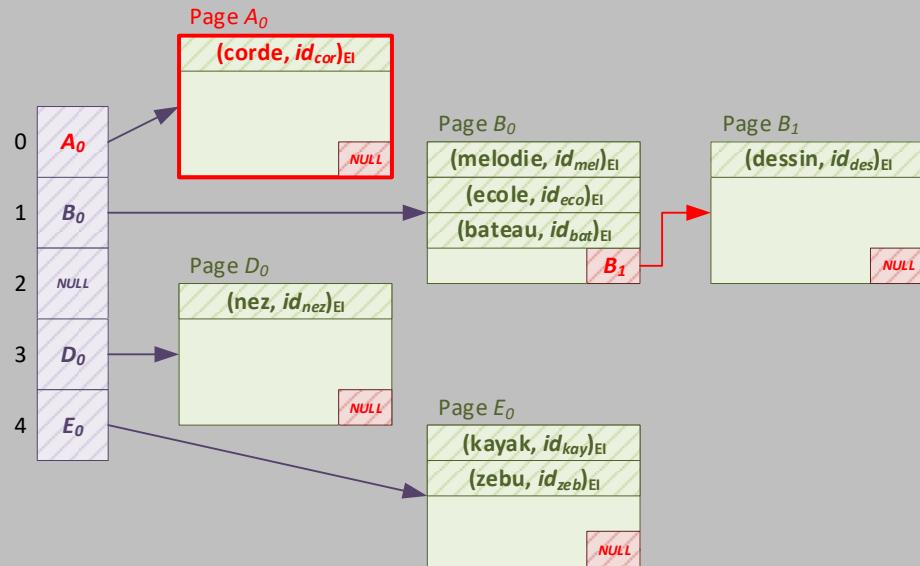


8. On ajoute dans la relation le mot dessin, donc l'entrée d'index de valeur dessin est insérée dans l'index : son code haché $h(\text{dessin})$ vaut 1 et il n'y a plus de place dans la page (unique pour le moment) « pointée » par la case d'indice 1 du répertoire de hachage statique. On crée donc une nouvelle page B_1 chaînée à la suite de B_0 et on ajoute l'entrée d'index dans cette nouvelle page.

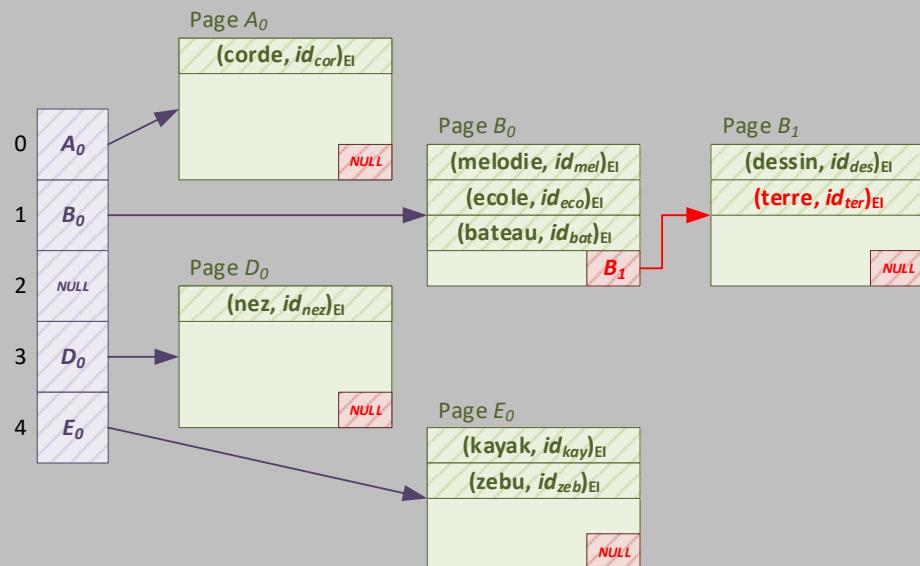


Exemple (suite)

9. On ajoute dans la relation le mot corde, donc l'entrée d'index de valeur corde est insérée dans l'index : son code haché $h(\text{corde})$ vaut 0 ; la case d'indice 0 du répertoire de hachage statique ne « pointant » vers aucune page, la page A_0 est créée, « pointée » depuis la case d'indice 0 et l'entrée d'index y est rajoutée.

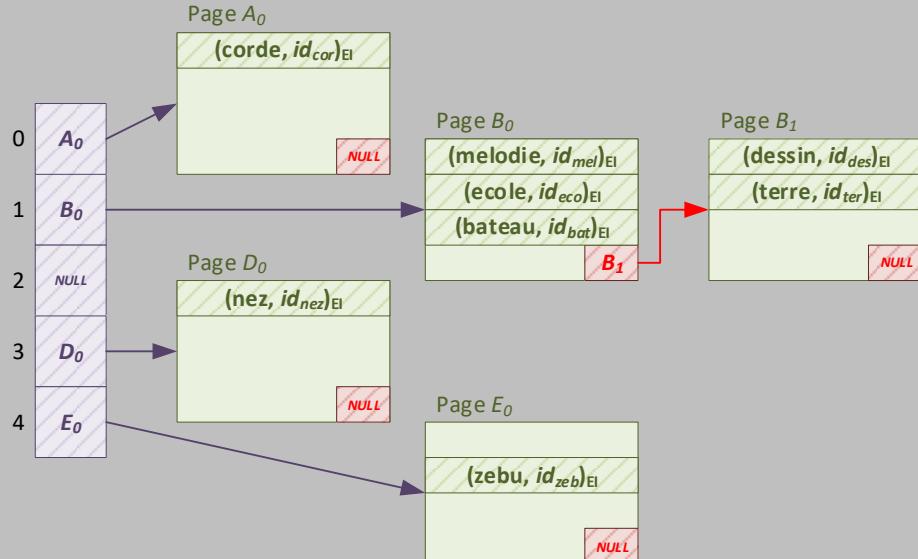


10. On ajoute dans la relation le mot terre, donc l'entrée d'index de valeur terre est insérée dans l'index : son code haché $h(\text{terre})$ vaut 1 et il y a de la place dans la liste chaîne de pages « pointée » par la case d'indice 1 du répertoire de hachage statique. On ajoute donc l'entrée d'index dans la première place libre de cette liste chaînée de pages.

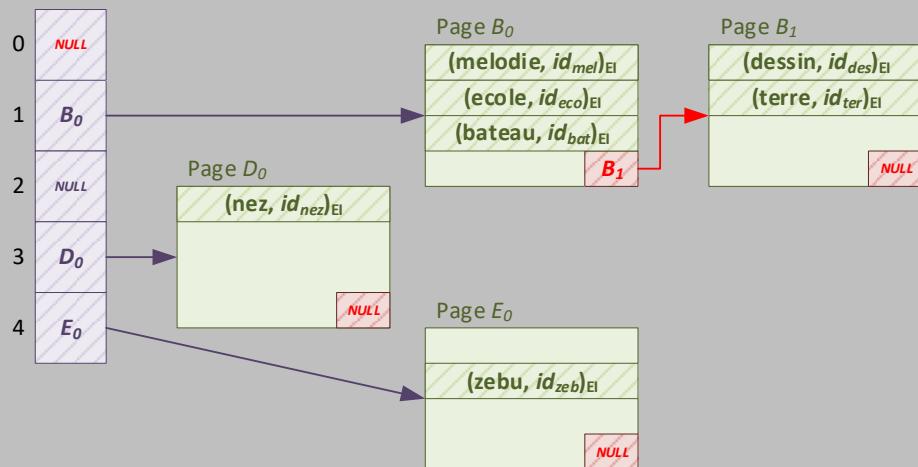


Exemple (fin)

11. On supprime de la relation le mot kayak, donc l'entrée d'index de valeur kayak est supprimée de l'index : son code haché $h(kayak)$ vaut 4 ; on la supprime donc de la liste chaînée de pages « pointée » par la case d'indice 4 du répertoire de hachage statique. La page de laquelle la suppression est faite ne devient pas vide, on ne modifie donc rien d'autre.



12. Enfin, on supprime de la relation le mot corde, dont l'entrée d'index de valeur corde est supprimée de l'index : son code haché $h(corde)$ vaut 0 ; on la supprime donc de la liste chaînée de pages « pointée » par la case d'indice 0 du répertoire de hachage statique. La page de laquelle la suppression est faite devient vide, on la supprime donc (et on met à jour ce qui « pointait » vers elle, i.e. la case d'indice 0 du répertoire de hachage statique) avec ce vers quoi elle « pointait » (i.e. rien !).





Remarque

Contrairement à celle d'un arbre B+, l'évolution d'un IMCAHSaR est facile à appréhender. On aurait pu directement trouver le dernier état de l'exemple ci-dessus simplement en sachant l'ordre des insertions/suppressions.

8.2.2.2.1.3 Performances d'un IMCAHSaR

Comme d'habitude, les performances dépendent principalement du nombre de transferts de pages à faire lors de l'usage d'un index mono-critère à accès par hachage statique avec répertoire. N'oublions pas que, dans cet index, on trouve :

- *Un répertoire de hachage statique R* : celui-ci est stocké dans des pages ne contenant que lui (elles ne contiennent en fait que des cases de ce répertoire de hachage statique et rien d'autre, **une case ne pouvant PAS être stockée à cheval sur plusieurs pages**),
- *Un ensemble de pages P* : elles ne contiennent quasiment que des entrées d'index (une entrée d'index ne pouvant PAS être stockée à cheval sur plusieurs pages) et un « pointeur » vers leur sœur droite en fin de page.

Donc, minimiser le coût d'usage de l'index passe implicitement par un remplissage optimal de ces sortes de pages. Il convient donc :

- *De remplir le plus possible les pages dans lesquelles sont stockées les cases du répertoire de hachage statique R* : cela ne pose aucun souci puisque le répertoire de hachage statique, donc son nombre de cases, est connu dès la création de l'index et ne varie pas,
- *De remplir au mieux les pages P contenant les entrées d'index* : on fait « au mieux », c'est d'ailleurs pour cela qu'on place une entrée d'index à la première place libre susceptible de l'accueillir, sans chercher à trier quoi que ce soit.



Remarque

Pour tenter d'améliorer le remplissage des pages contenant les entrées d'index (donc minimiser leur nombre), on peut éventuellement se dire qu'on peut mettre en œuvre une « défragmentation » de chaque liste chaînée de pages. Pourquoi pas, mais il faut alors bien veiller à ce que la défragmentation ne coûte pas plus cher que le gain qu'elle va apporter par la suite.

Avec un bon ajustement de $h()$ ¹⁷⁷, donc de la valeur de N , et du nombre d'entrées d'index par page, on peut accéder à une entrée d'index en 1,1 ou 1,2 accès disque en moyenne, si le répertoire de hachage statique tient en mémoire de travail¹⁷⁸, quel que soit le nombre d'entrées d'index !

¹⁷⁷ Toute la difficulté étant bien sûr là, vous l'aurez compris !

¹⁷⁸ Il est bien sûr tout à fait possible d'épingler dans la mémoire cache les pages qui le contiennent (si le gestionnaire de mémoire cache supporte le punaisage des pages évidemment, cf. §7.3.4) !

Les paramètres propres à un IMCAHSaR sont les suivants :

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$h(x)$ et N	-	Fonction de hachage telle que $0 \leq h(x) < N$
$nbMax_{entrees/page}^{fixe moy min max}(I(R, c))$	entrées d'index par page	<p>Nombre (fixe ou moyen ou minimal ou maximal) d'entrées d'index que l'on peut stocker au plus dans une page d'un IMCAHSaR :</p> $nbMax_{entrees/page}^{fixe moy min max}(I(R, c)) \triangleq \frac{T_{page}^{fixe} (*) - T_{idPage}^{fixe}}{T_{entree}^{fixe moy min max}(I(R, c))}$
$nb_{pagesEntrees}^{fixe moy min max}(I(R, c))$	pages	<p>Nombre (fixe ou moyen ou minimal ou maximal) de pages contenant les entrées d'index :</p> $nb_{pagesEntrees}^{fixe moy min max}(I(R, c)) \triangleq \frac{nb_{entrees}^{exact}(I(R, c))}{nbMax_{entrees/page}^{fixe moy min max}(I(R, c))}$
$T_{caseRepHS}^{fixe}(I(R, c))$	octets	<p>Taille (forcément fixe) d'une case du répertoire de hachage statique :</p> $T_{caseRepHS}^{fixe}(I(R, c)) = T_{idPage}^{fixe}(*)$
$nb_{casesRepHS}^{fixe}(I(R, c))$	octets	<p>Nombre (forcément fixe) de cases dans le répertoire de hachage statique :</p> $nb_{casesRepHS}^{fixe}(I(R, c)) = N$
$nbMax_{casesRepHS/page}^{fixe}(I(R, c))$	cases par page	<p>Nombre maximal (forcément fixe) de cases du répertoire de hachage statique au plus par page :</p> $nbMax_{casesRepHS/page}^{fixe}(I(R, c)) \triangleq \frac{T_{page}^{fixe} (*)}{T_{caseRepHS}^{fixe}(I(R, c))}$
$nb_{PagesRepHS}^{fixe}(I(R, c))$	pages	<p>Nombre (forcément fixe) de pages occupées par le répertoire de hachage statique :</p> $nb_{PagesRepHS}^{fixe}(I(R, c)) = \frac{nb_{casesRepHS}^{fixe}(I(R, c))}{nb_{casesRepHS/page}^{fixe}(I(R, c))}$
$T_{listePages}^{moy}(I(R, c))$	pages	<p>Taille (moyenne) des listes chaînées de pages dans lesquelles les entrées d'index sont stockées :</p> $T_{listePages}^{moy}(I(R, c)) \triangleq \frac{nb_{pagesEntrees}^{fixe moy min max}(I(R, c))}{N}$
$Coût_{usage}^{moy}(I(R, c))$	transferts de pages (en lecture)	<p>Coût (moyen) d'usage de l'IMCAHSaR :</p> $Coût_{usage}^{moy}(I(R, c)) = T_{listePages}^{moy}(I(R, c)) \stackrel{\text{si repHS}}{\underset{\text{non punaisé}}{\overset{+1}{\triangleq}}}$

Tableau 45. Performances d'indexation : paramètres spécifiques aux IMCAHSaR

8.2.2.2.2 Organisation SANS répertoire (IMCAHSsR)

Afin d'uniformiser, donc de faciliter, la gestion d'un index mono-critère à accès par hachage statique, on peut éviter l'utilisation d'un répertoire en « simulant » un : il suffit pour cela de stocker en mémoire de travail les premières pages de chaque liste chaînée de pages de façon contiguë et ordonnée (par ordre croissant du code haché associé à chacune de ces listes chaînées). Ce « bloc » des premières pages de chaque liste chaînée peut donc jouer le rôle d'un répertoire (bien que ça n'en soit donc pas un en réalité) : on peut en effet accéder à une page de ce « bloc » grâce à sa position (qui débute donc à 0 pour la 1^{ère}). Une page de ce « bloc » a donc une position au sein de celui-ci égale au code haché associé à toutes les entrées d'index qu'elle contient.



Question

Et pour « simuler » les cases du répertoire qui ne « pointaient » rien ?

Réponse

Simplement en les « remplaçant » par des pages vides.

Les pages suivantes des listes chaînées peuvent être placées « n'importe où » dans la mémoire de travail : cela ne pose aucun problème puisqu'on les atteint par leur identificateur de page.



Exemple (début)

Voici ci-dessous un index mono-critère à accès par hachage statique avec répertoire :

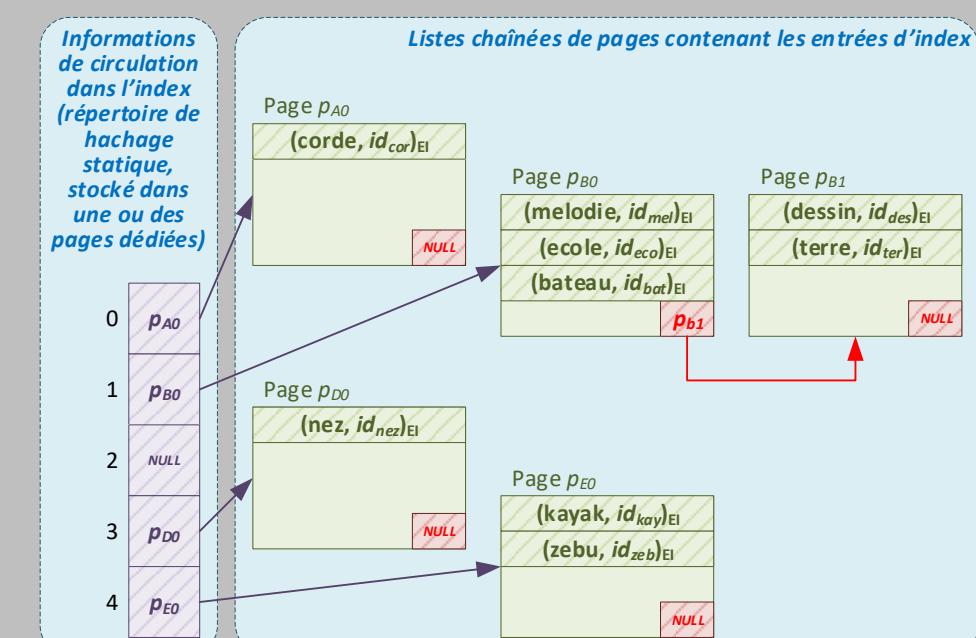


Figure 126. Exemple d'index mono-critère à accès par hachage statique avec répertoire

Exemple (fin)

Voici ci-dessous le même mais sans répertoire de hachage statique :

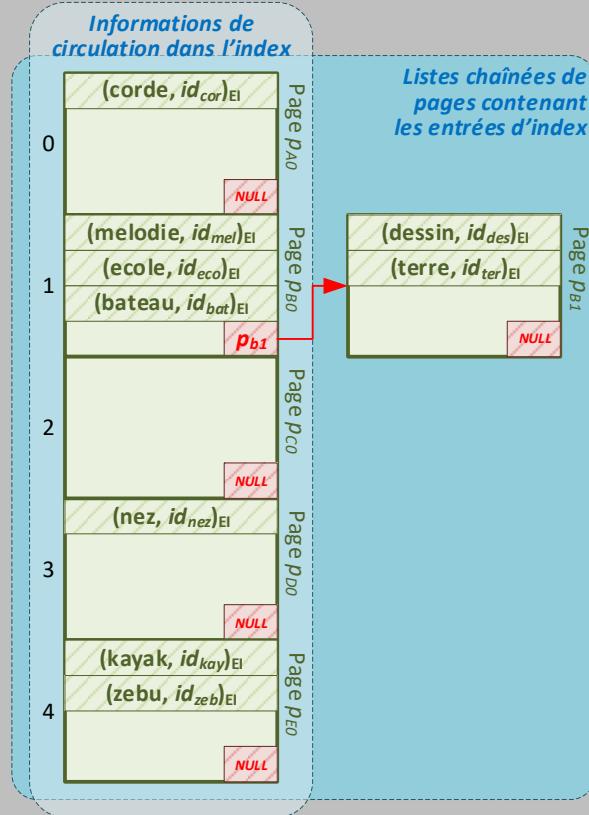


Figure 127. Exemple d'index mono-critère à accès par hachage statique sans répertoire

Définition : « index mono-critère à accès par hachage statique sans répertoire »

Un **index mono-critère à accès par hachage statique sans répertoire** est une structure d'index adossée à une fonction de hachage $h()$ telle que $0 \leq h(x) < N$ (où N dépend donc de la fonction de hachage $h()$ choisie). Cette structure d'index prend la forme d'un ensemble de pages P organisées en plusieurs listes chaînées de pages (une liste chaînée par code haché possible). Les premières pages de chaque liste chaînée sont stockées de façon contigüe en mémoire de travail, en « bloc », ce « bloc » jouant le rôle d'un répertoire et la position d'une première page de liste chaînée dans ce « bloc » étant égale au code haché qui lui est associé (la première position est 0). Toutes les entrées d'index (v, a) ayant le même code haché $h(v) = i$ se trouvent dans la liste chaînée de pages initiée par la page en position i dans le « bloc » de pages de tête des listes chaînées.

Ainsi, avec une organisation sans répertoire, aucune information de circulation particulière n'est donc directement nécessaire : on ne gère finalement que des pages de stockage des entrées d'index, il suffit simplement de stocker de façon contigüe certaines d'entre elles : les premières de chaque liste chaînée de pages associée à chaque code haché « possible », qui jouent donc le rôle du répertoire.

Au niveau des opérations à mettre en œuvre pour une organisation sans répertoire, il ne s'agit que de variantes des opérations décrites plus haut pour une organisation avec répertoire. Rien de bien nouveau donc... Enfin, au niveau des performances, toujours avec un bon ajustement de $h()$ ¹⁷⁹, donc de la valeur de N , et du nombre d'entrées d'index par page, on peut accéder à une entrée d'index en 1,1 ou 1,2 accès disque en moyenne, quel que soit le nombre d'entrées d'index !

Enfin, on peut globalement voir comme suit les différentes structures de pages intervenant dans le stockage d'un IMCAHS sans répertoire et dans celui de la relation indexée par cet IMCAHSsR :

- Si la relation indexée est en mode d'adressage direct :

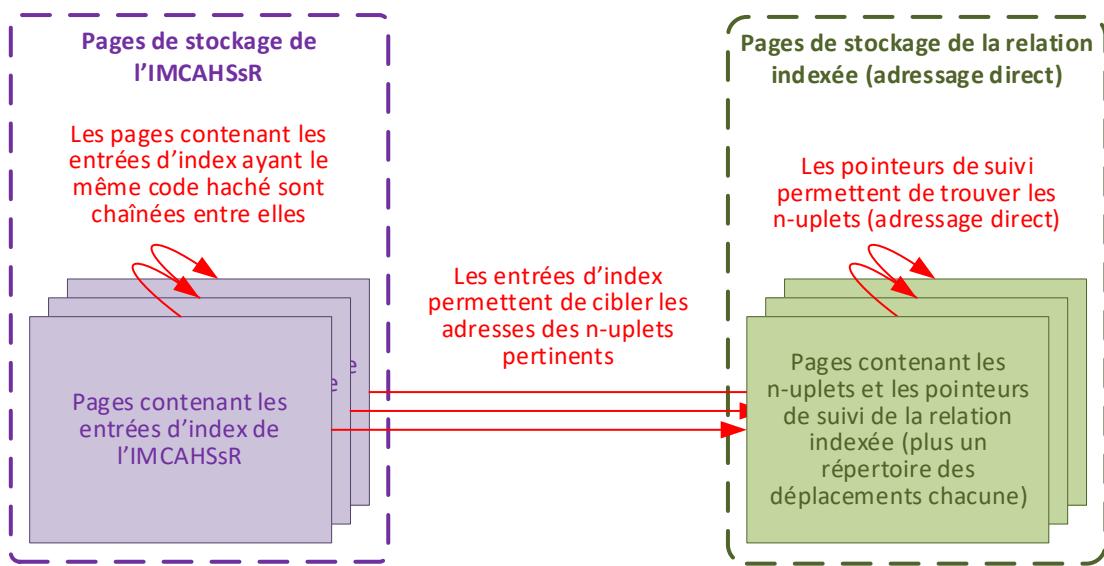


Figure 128. Pages de stockage d'un IMCAHSsR et de la relation indexée (adressage direct)

¹⁷⁹ Toute la difficulté étant bien sûr là, vous l'aurez compris !

- Si la relation indexée est en mode d'adressage indirect :

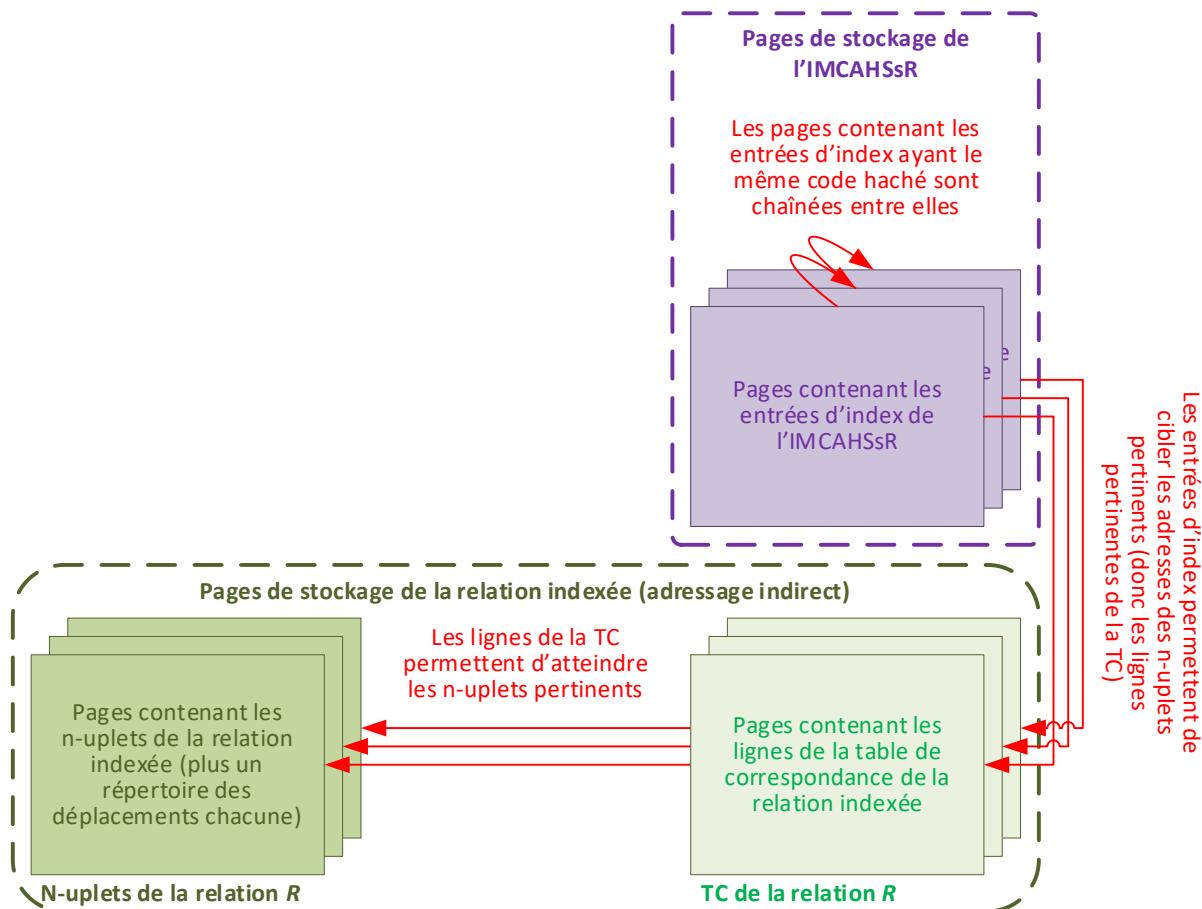


Figure 129. Pages de stockage d'un IMCAHSsR et de la relation indexée (adressage indirect)

8.2.2.2.3 Discussion « AVEC » vs « SANS » répertoire

Si les 2 organisations co-existent encore, c'est qu'elles présentent chacune des avantages et des inconvénients... Ainsi, lorsqu'il n'existe pas d'entrée d'index valant un certain code haché i , la place perdue dans le cas de l'organisation avec répertoire n'est que celle de la case d'indice i du répertoire (i.e. la place occupée par un identificateur de page) alors que dans le cas de l'organisation sans répertoire, la place perdue est celle de tout une page.

De plus, dans le cas de l'organisation sans répertoire, on ne gère que des pages, donc que des « éléments » qui ont tous la même structure alors que dans le cas d'organisations avec répertoire, il faut gérer des pages contenant des entrées d'index donc avec une certaine structure interne) et des pages contenant des cases du répertoire (donc avec une autre structure interne). Enfin, dans le cas de l'organisation avec répertoire, si celui-ci ne tient pas entièrement en mémoire de travail, un accès disque supplémentaire au moins est nécessaire (on passe donc à 2,1 ou 2,2 accès disque en moyenne).

Organisation de l'IMCAHS	Avantages	Inconvénients
Avec répertoire	Peu de perte de place en cas de codes hachés $h(v)$ « perdus » (une case du répertoire pour chacun)	Deux types de pages à gérer (celles qui contiennent les cases du répertoire et celles qui contiennent les entrées d'index)
Sans répertoire	Facilité de gestion (un seul type de pages à gérer)	Perte de place importante en cas de codes hachés $h(v)$ « perdus » (une page pour chacun).

Tableau 46. Bilan des organisations d'IMCAHS

Les paramètres liés aux IMCAHSsR sont similaires à ceux que l'on trouve dans l'organisation avec répertoire : seuls ceux liés au répertoire de hachage statique ne sont pas pris en compte (puisque il n'y en a tout simplement pas ici !).

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$h(x)$ et N	-	Fonction de hachage telle que $0 \leq h(x) < N$
$nbMax_{entrees/page}^{fixe moy min max}(I(R, c))$	entrées d'index par page	Nombre (fixe ou moyen ou minimal ou maximal) d'entrées d'index que l'on peut stocker au plus dans une page d'un IMCAHSaR : $nbMax_{entrees/page}^{fixe moy min max}(I(R, c)) \leq \frac{T_{page}^{fixe} (*) - T_{idPage}^{fixe}}{T_{entree}^{fixe moy min max}(I(R, c))}$
$nb_{pagesEntrees}^{fixe moy min max}(I(R, c))$	pages	Nombre (fixe ou moyen ou minimal ou maximal) de pages contenant les entrées d'index : $nb_{pagesEntrees}^{fixe moy min max}(I(R, c)) = \frac{nb_{entrees}^{exact}(I(R, c))}{nbMax_{entrees/page}^{fixe moy min max}(I(R, c))}$
$T_{listePages}^{moy}(I(R, c))$	pages	Taille (moyenne) des listes chaînées de pages dans lesquelles les entrées d'index sont stockées : $T_{listePages}^{moy}(I(R, c)) = \frac{nb_{pagesEntrees}^{fixe moy min max}(I(R, c))}{N}$
$Coût_{usage}^{moy}(I(R, c))$	transferts de pages (en lecture)	Coût (moyen) d'usage de l'IMCAHSaR : $Coût_{usage}^{moy}(I(R, c)) = T_{listePages}^{moy}(I(R, c))$

Tableau 47. Performances d'indexation : paramètres spécifiques aux IMCAHSaR

8.2.2.3 Index mono-critère à accès par hachage dynamique (IMCAHD)

Un des plus gros inconvénients des techniques de hachage statique réside, précisément, dans la permanence de la fonction de hachage $h()$: celle-ci est choisie à la définition de l'index et reste la même durant toute la vie de cet index. Il est donc très fréquent que la fonction de hachage soit « sous-dimensionnée » à des moments de la vie de l'index (quand il contient beaucoup d'entrées d'index en regard du maximum N retourné par la fonction de hachage choisie) mais aussi « surdimensionnée » à d'autres moments (quand l'index contient peu d'entrées d'index en regard du maximum N retourné par la fonction de hachage choisie). Tout revient à un problème de collisions dans un cas où à un problème de place perdue dans l'autre :

- *Quand $h()$ est « sous-dimensionnée »* : le nombre de collisions augmente inexorablement au fur et à mesure que l'index se remplit, donc les performances de l'index se dégradent,
- *Quand $h()$ est « surdimensionnée »* : de la place est perdue en raison de toutes les valeurs de code hachés « inutilisées ».



Remarque

On touche bien du doigt ici la difficulté de bien choisir sa fonction de hachage $h()$. Pour bien faire, il est nécessaire d'essayer d'estimer *a priori* le volume d'entrées d'index que l'on n'est susceptible d'avoir « en rythme de croisière », ce qui revient donc à essayer d'estimer *a priori* le volume de valeurs différentes du constituant indexé que l'on est susceptible d'avoir dans les n-uplets de la relation « en rythme de croisière »... Pas simple !

Une solution consisterait à faire évoluer la taille du répertoire, donc la fonction de hachage $h()$, en fonction du nombre courants d'entrées d'index dans l'index¹⁸⁰. Cette idée a donné naissance à la technique du **hachage dynamique** (ou **hachage extensible**) introduite par Fagin *et al.* en 1979. De nombreuses variantes ont été introduites depuis, on présente la plus courante ci-dessous...

8.2.2.3.1 Organisation

Un index mono-critère à accès par hachage dynamique est défini par :

- Une **profondeur de répertoire** d , variable (c'est un entier naturel valant initialement 0),
 - Une **fonction de hachage** $h()$, qui associe à une valeur v un code haché $h(v)$ tel que $0 \leq h(v) < N$.
- Ce code haché n'est pas utilisé directement** : une **pseudo-clé** est calculée en convertissant en entier naturel les d bits de poids forts du codage binaire du code haché $h(v)$ (donc l'usage fait des codes hachés pour obtenir les pseudo-clés évolue avec d),
- Un **répertoire de hachage dynamique** R , dont le nombre de cases est égal à 2^d (ce nombre va donc varier dans le temps en même temps que d), ces cases étant indiquées de 0 à $2^d - 1$ ¹⁸¹.
 - Un **ensemble de pages**, P , « pointées » depuis le répertoire de hachage dynamique R et contenant les entrées d'index. Chacune de ces pages est caractérisée par une **profondeur locale** p (qui lui est propre, donc) : c'est un entier naturel (avec $0 \leq p \leq d$) indiquant le nombre de bits de poids fort qu'ont forcément en commun les pseudo-clés de toutes les entrées d'index situées dans la page¹⁸².

¹⁸⁰ Nous allons le voir, en réalité la « fonction de hachage » n'évoluera pas ! Ce qui évoluera, en revanche, c'est bien l'interprétation (*i.e.* l'usage) qui sera fait des résultats qu'elle retourne et, simultanément, la taille du répertoire (et c'est bien ce dernier point qu'on voulait précisément réaliser).

¹⁸¹ Puisqu'on utilise les d bits de poids fort des codes hachés $h(v)$ pour former les pseudo-clés et qu'on peut coder sur d bits 2^d entiers naturels, le répertoire de hachage dynamique contient bien 2^d cases indiquées de 0 à $2^d - 1$.

¹⁸² Si ça n'est pas le cas, c'est qu'une erreur a été faite pendant la « construction » de l'index.



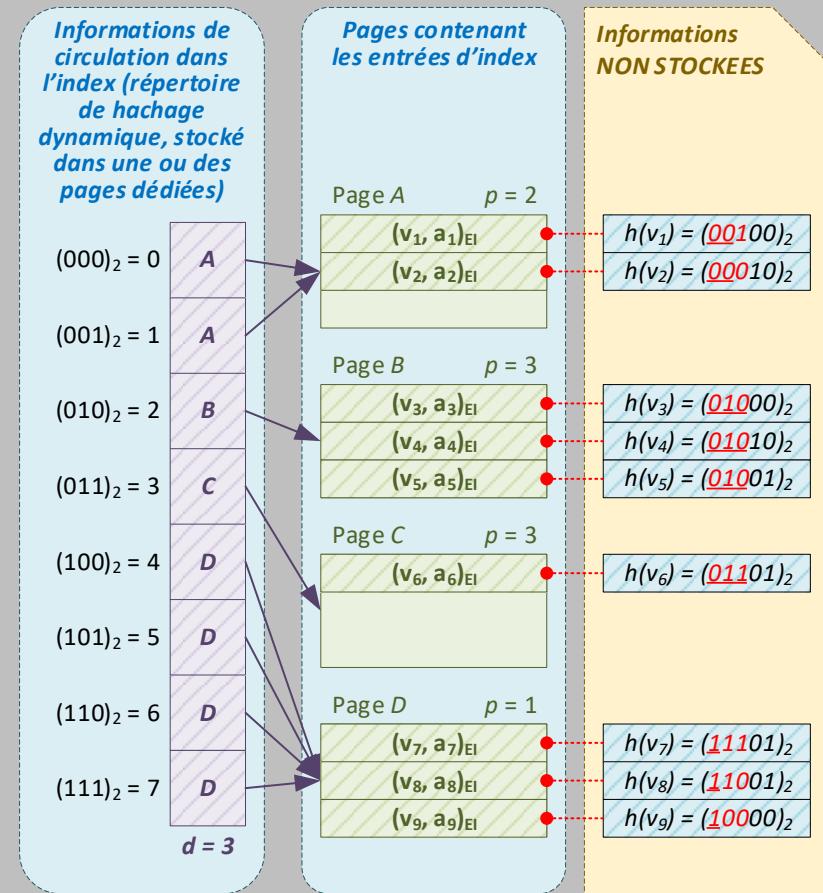
Attention

Un index mono-critère à accès par hachage dynamique a TOUJOURS un répertoire : il n'existe pas d'organisation équivalente sans répertoire !



Exemple

Voici une « illustration » de l'organisation générale d'un index mono-critère à accès par hachage dynamique :



Ces informations sont données uniquement pour la compréhension de l'exemple : ce sont les codes hachés $h(v)$. Les bits en rouge sont les d bits de poids fort de ces codes hachés et les bits soulignés (forcément rouges aussi) sont les p bits de poids fort communs aux codes hachés $h(v)$ des entrées d'index de valeur v situés dans une même page.

Figure 130. Organisation générale d'un index mono-critère à accès par hachage dynamique



Remarque

Notez que, dans le cas d'un index mono-critère à accès par hachage dynamique, les pages contenant les entrées d'index n'ont pas de sœurs vers lesquelles « pointer ». C'est un effet de la bonne répartition des collisions faite dans ce type d'index : on n'a jamais de listes chaînées de 2 pages ou plus associées à une case du répertoire.

**Astuce**

À tout instant, la profondeur locale p d'une page est reliée à la profondeur du répertoire d et au nombre n de cases de ce répertoire qui pointent vers elle : par construction d'un index à accès par hachage dynamique, n s'exprime forcément sous la forme 2^i et on a forcément à tout instant pour tout page $p = d - i^{183}$.

L'idée de base est la même que pour le hachage statique : une entrée d'index (v, a) se trouve dans une page « pointée » par la case du répertoire d'indice i . La valeur de cet indice résulte de l'interprétation comme un entier naturel des d bits de poids fort de la pseudo-clé formée à partir de $h(v)$. Par contre, avec un hachage dynamique, on va pouvoir faire évoluer la valeur de d (donc les pseudo-clés et le nombre de cases du répertoire de hachage dynamique R) au besoin :

- *Quand le nombre d'entrées d'index grandit* : on finit par incrémenter la valeur de la profondeur du répertoire d de 1. Cela a pour effet de doubler le nombre de cases dans le répertoire de hachage dynamique et de re-répartir les entrées d'index (puisque le nombre de bits de poids fort pris en compte dans les codes hachés $h(v)$ pour former les pseudo-clés évolue lui aussi), donc de réduire les collisions !
- *Quand le nombre d'entrées d'index diminue* : on finit par décrémenter la valeur de la profondeur du répertoire d de 1. Cela a pour effet de diviser par 2 le nombre de cases dans le répertoire de hachage dynamique et de re-répartir les entrées d'index (puisque, là encore, le nombre de bits de poids fort pris en compte dans les codes hachés $h(v)$ pour former les pseudo-clés évolue en même temps), donc de réduire la place « perdue ».

Une première question peut d'ores et déjà venir à l'esprit...

**Question**

Puisque les codes hachés $h(v)$ ne sont pas directement utilisés (seuls leurs d bits de poids fort le sont), comment bien choisir la fonction de hachage $h()$ pour que le hachage soit performant ?

Réponse

Ici, cela n'a en fait que peu d'importance ! C'est le mécanisme-même d'évolution des index mono-critères à accès par hachage dynamique qui veille à ce que les collisions soient réparties de façon équiprobables : ça n'est donc pas directement le choix de la fonction de hachage $h()$. En revanche, il vaut mieux choisir cette fonction de hachage $h()$ telle que $0 \leq h(v) < N$ avec N grand (voire très grand) ! Vu qu'on utilise les d bits de poids fort des codes hachés et que d peut varier, et donc devenir « grand » si besoin, autant que les codes hachés soient codés (en binaire) sur beaucoup de bits, au cas où... Si ça n'est pas le cas (et que d reste « petit »), ça n'est pas grave étant donné que les codes hachés ne sont pas stockés.

¹⁸³ Donc, vérifier cela pendant la construction d'un tel index permet de voir si l'on s'est trompé. En revanche, l'inverse n'est pas vrai : retrouver ce lien à tout instant ne suffit pas à assurer que l'index est correct (c'est donc nécessaire mais pas suffisant).

Comme dans le cas des index mono-critère à accès par hachage dynamique avec répertoire (cf. §8.2.2.2.1), on peut globalement voir comme suit les différentes structures de pages intervenant dans le stockage d'un IMCAHD sans répertoire et dans celui de la relation indexée par cet IMCAHD :

- Si la relation indexée est en mode d'adressage direct :

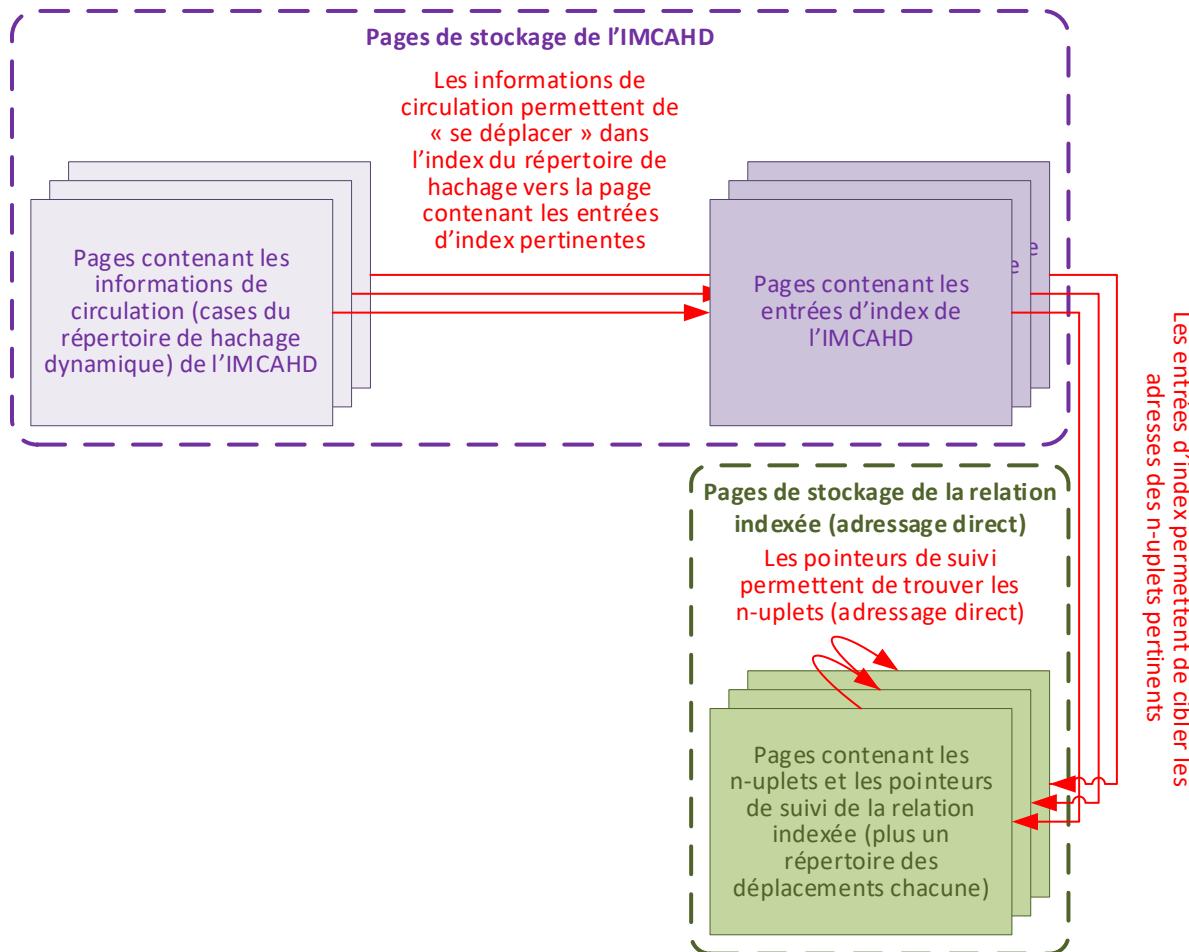


Figure 131. Pages de stockage d'un IMCAHD et de la relation indexée (adressage direct)

- Si la relation indexée est en mode d'adressage indirect :

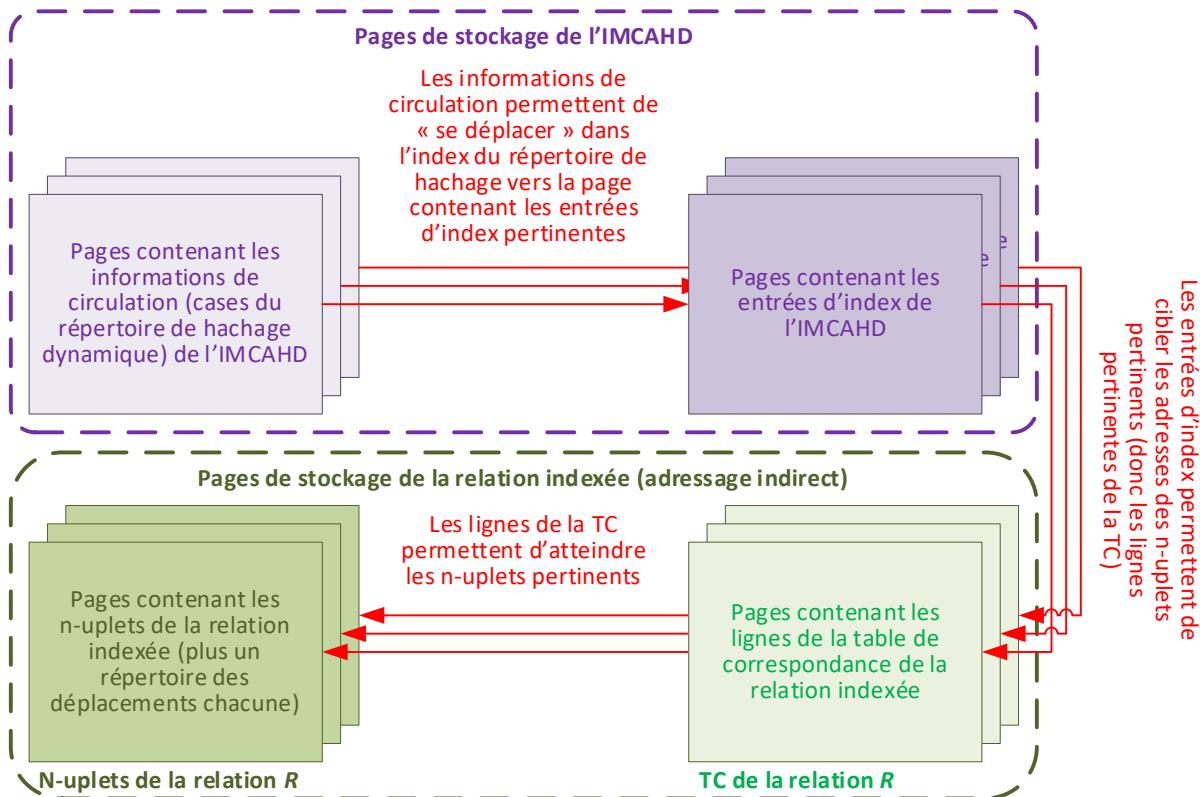


Figure 132. Pages de stockage d'un IMCAHD et de la relation indexée (adressage indirect)

8.2.2.3.2 Évolution d'un IMCAHD : opérations de manipulation

Encore une fois, un index (donc un IMCAHD) vit en fonction de l'évolution de la relation qu'il indexe. À sa création, un index mono-critère à accès par hachage dynamique est défini par une profondeur du répertoire nulle ($d = 0$). Son répertoire de hachage dynamique n'a donc qu'une seule case et ne contient rien. L'ajout de la première entrée d'index provoque la création d'une page de profondeur locale nulle ($p = 0$) et la suppression de la dernière entrée d'index provoquera la suppression de cette dernière page. Les ajouts suivants provoqueront éventuellement des divisions de pages voire l'augmentation de la profondeur du répertoire d (donc le doublement de la taille du répertoire de hachage dynamique). À l'inverse, les suppressions d'entrées d'index provoqueront éventuellement des fusions de pages voire la diminution de la profondeur du répertoire d (donc la division par 2 de la taille du répertoire de hachage dynamique).

Les principales opérations réalisables sur un index par hachage dynamique sont :

- La recherche d'une entrée d'index,
- L'insertion d'une entrée d'index,
- La division d'une page en deux,
- L'agrandissement (doublement) du répertoire de hachage dynamique ($d \leftarrow d + 1$),
- La suppression d'une entrée d'index (qu'on ne verra pas ici),
- La fusion de pages (qu'on ne verra pas ici),
- La réduction du répertoire de hachage dynamique ($d \leftarrow d - 1$, qu'on ne verra pas ici).

8.2.2.3.2.1 Recherche d'une entrée d'index

La recherche d'une entrée d'index (v, a) se fait comme suit : on regarde le codage binaire de la pseudo-clé, *i.e.* les d bits de poids fort du code haché $h(v)$, et on l'interprète comme un entier naturel i . Ce nombre i est l'indice de la case du répertoire de hachage dynamique « pointant » vers la page contenant l'entrée d'index recherchée (si elle existe).



Question

Et si d vaut 0, la pseudo-clé est formée des « 0 bits de poids fort » des codes hachés ?

Réponse

Effectivement, cela n'a pas de sens. Si d vaut 0, on se place automatiquement dans la case d'indice 0 du répertoire : par construction, c'est de toutes les façons la seule case qu'il contient !



En pratique

Le pseudo-algorithme suivant illustre la recherche d'une entrée d'index.

```
Pseudo-algorithme RechercheEIIMCAHD
Données d'entrée
    v : une valeur
Données de sortie
    EI : l'entrée d'index (v, a) recherchée
    p : l'identificateur de la page contenant EI
Données intermédiaires
    i : l'indice d'une case du répertoire de l'IMCAHD
    h() : la fonction de hachage de l'IMCAHD
    d : la profondeur du répertoire
Début
    Si d = 0 alors
        i ← 0
    Sinon
        Calculer h(v)
        i ← pseudo-clé (d bits de poids fort de h(v))
    FinSi
    p ← id. de page contenu dans la case i du répertoire
    Si p = NULL alors
        EI ← NULL
    Sinon
        Chercher EI dans la page p
        Si EI n'existe pas dans la page p alors
            EI ← NULL
        FinSi
    FinSi
Fin
```

8.2.2.3.2.2 Insertion d'une entrée d'index

L'insertion d'une entrée d'index (v, a) se fait classiquement en cherchant la page où cette entrée devrait se trouver. S'il y a assez de place, on l'y insère. Sinon, on divise cette page en 2 puis on cherche de nouveau à réaliser l'insertion. Comme d'habitude également, s'il existe déjà une entrée d'index de valeur v dans l'index, on concatène à sa liste d'adresses logiques celle qui est contenue dans l'entrée d'index que l'on cherche à insérer.

En pratique

Le pseudo-algorithme suivant montre le fonctionnement de cette opération.

```

Pseudo-algorithme InsertionEIIMCAHD
Données d'entrée
    EIins : l'entrée d'index ( $v, a$ ) à insérer
Données de sortie
    p : page dans laquelle l'insertion est réalisée
Données intermédiaires
    EIch : une entrée d'index
    i : l'indice d'une case du répertoire de l'IMCAHD
    h() : la fonction de hachage de l'IMCAHD
    profrep : la profondeur du répertoire
    profloc : la profondeur locale d'une page
    insertion : un drapeau
Début
    insertion ← FAUX
Répéter
    Rechercher l'entrée d'index de valeur  $v$ 184
    EIch ← entrée d'index trouvée
    p ← id. de page retourné par la recherche
    Si EIch existe alors
        Concaténer a aux adresses logiques dans EIch
        insertion ← VRAI
    Sinon
        profloc ← profondeur locale de la page p
        Si p a de la place pour accueillir EIins alors
            Insérer EIins dans p
            insertion ← VRAI
        sinon
            Si profloc < profrep alors
                Diviser la page p en deux185
            Sinon
                Doubler le répertoire de l'IMCAHD186
            FinSi
        FinSi
    Jusqu'à insertion = VRAI
Fin
```



¹⁸⁴ Voir le pseudo-algorithme RechercheEIIMCAHD plus haut (cf. §8.2.2.3.2.1).

¹⁸⁵ Voir le pseudo-algorithme DivisionPageIMCAHD plus bas (cf. §8.2.2.3.2.3).

¹⁸⁶ Voir le pseudo-algorithme DoublementRepIMCAHD plus bas (cf. §8.2.2.3.2.4).

8.2.2.3.2.3 Division d'une page (en deux)

Cette opération revient à créer une nouvelle page et à affecter aux 2 (la page d'origine et la nouvelle) une profondeur locale p égale à la profondeur locale p d'origine de la page divisée incrémentée de 1. C'est en fonction de ces nouvelles profondeurs locales que les entrées d'index contenues dans la page d'origine vont être réparties sur les 2 pages issues de la division.

En pratique

Le pseudo-algorithme ci-dessous illustre le comportement attendu :

Pseudo-algorithme DivisionPageIMCAHD

Données d'entrée

p : la page d'origine à diviser

Données de sortie

p : la page d'origine une fois la division faite

p' : la nouvelle page issue de la division

Données intermédiaires

$prof_{loc}$: la profondeur locale d'une page

k : le nombre de cases du répertoire « pointant » p

Début

$prof_{loc} \leftarrow$ profondeur locale de p

$p' \leftarrow$ nouvelle page vide

Modifier la profondeur locale de p en $prof_{loc} + 1$

Affecter $prof_{loc} + 1$ à la profondeur locale de p'

Transférer dans p' toutes les entrées d'index de p

dont le $prof_{loc}$ ème bit de poids fort de leur

pseudo-clé vaut 1 (les autres entrées d'index restent dans p)

Mettre à jour le répertoire de la façon suivante :

soit k le nombre de cases du répertoire pointant vers la page initiale ; faire pointer vers la nouvelle page p' les $k / 2$ dernières de ces cases.

Fin



8.2.2.3.2.4 Doublement du répertoire de hachage dynamique

Le doublement du répertoire de hachage dynamique permet de prendre en compte l'insertion de nombreuses entrées d'index. Cette opération consiste simplement à dédoubler ses cases.



Attention

Il ne s'agit SURTOUT PAS de copier le répertoire de hachage dynamique sous lui-même ! Chacune de ses cases « donne naissance » à 2 cases adjacentes de même contenu dans le répertoire de hachage dynamique issu de l'agrandissement (cf. Figure 133). Pour en être persuadé, essayez de raisonner sur l'effet de l'incrément de d sur les indices des cases (plus précisément sur le codage binaire de ces indices)...

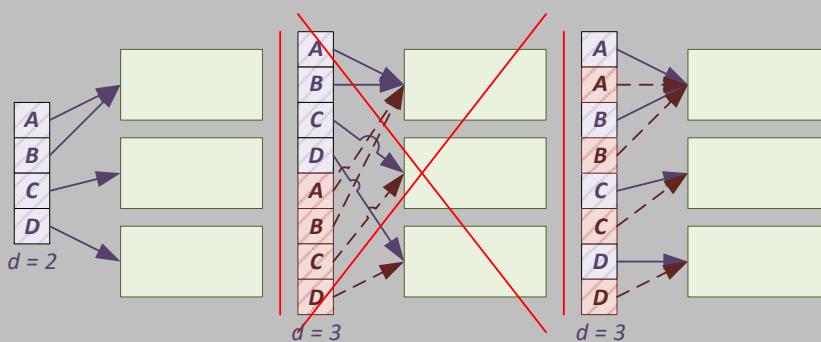


Figure 133. Mauvais et bon résultat du doublement du répertoire de hachage dynamique



En pratique

Le pseudo-algorithme suivant illustre ce doublement :

```

Pseudo-algorithme DoublementRepIMCAHD
Données d'entrée
    R : le répertoire d'un IMCAHD
Données de sortie
    R : le répertoire après agrandissement
    d : la profondeur du répertoire
Données intermédiaires
    c : une case du répertoire R
    c' : une case du répertoire R après accroissement
    s : case suivant initialement c dans R
    p : un identificateur de page
Début
    c ← première case du répertoire R
Répéter
    s ← case suivant c dans le répertoire R
    c' ← nouvelle case
    Insérer c' dans R entre c et sa suivante s
    p ← id. de page contenu dans c
    Écrire p dans la nouvelle case c'
    c ← s
Jusqu'à avoir traité chaque case initiale de R
    d ← d + 1
Fin
```

8.2.2.3.2.5 Exemple de manipulation d'un IMCAHD

On se propose de montrer l'évolution d'un index mono-critère à accès par hachage dynamique au fur et à mesure de l'ajout/suppression de n-uplets à/de la relation indexée. On suppose que la relation indexée est la relation Dictionnaire (cf. Tableau 37), que cette relation est initialement vide et que l'index a la structure d'un index mono-critère à accès par hachage dynamique. On suppose que, dans cet index, les pages de l'ensemble de pages P peuvent contenir au maximum 3 entrées d'index.

Exemple (début)

On commence par donner ci-dessous les 3 bits de poids fort des codes hachés des mots qui vont être ajoutés au dictionnaire (dans l'ordre dans lequel ces mots vont être ajoutés) :

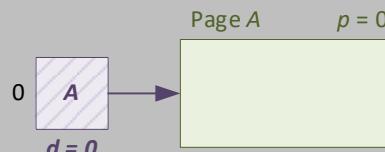
Mot (valeur v)	3 bits de poids fort des pseudo-codes hachés ($h(v)$)
melodie	001
ecole	100
nez	010
bateau	111
kayak	101
zebu	000
dessin	011
corde	110

Tableau 48. Bits de poids fort des codes hachés utilisés pour l'exemple d'IMCAHD

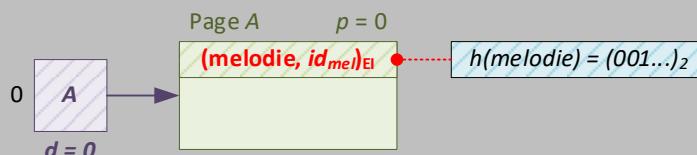


La relation va évoluer opération après opération, l'index associé également :

1. Initialement, la relation Dictionnaire est vide, l'index aussi : on démarre avec $d = 0$ donc avec 1 seule case dans le répertoire de hachage dynamique (d'indice 0), case qui pointe vers une unique page vide.

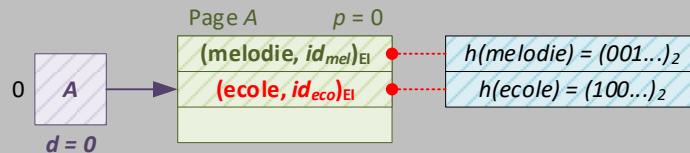


2. On ajoute dans la relation le mot `melodie`, donc l'entrée d'index associée à la valeur `melodie` est insérée dans l'index : d vaut 0, donc on « se dirige » automatiquement vers la seule case du répertoire de hachage dynamique. Elle « pointe » vers la page A . Celle-ci pouvant recevoir l'entrée d'index, on l'y insère.

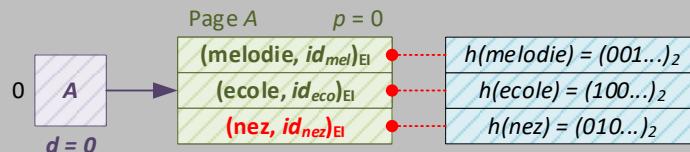


Exemple (suite)

3. On ajoute dans la relation le mot `ecole`, donc l'entrée d'index associée à la valeur `ecole` est insérée dans l'index : d vaut 0, donc on « se dirige » automatiquement vers la seule case du répertoire de hachage dynamique qui « pointe » vers la page A. Celle-ci pouvant recevoir l'entrée d'index, on l'y insère.



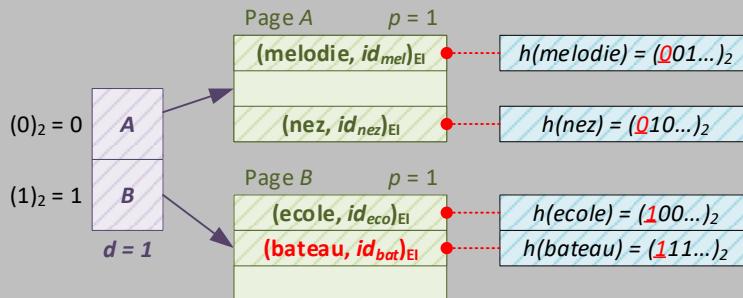
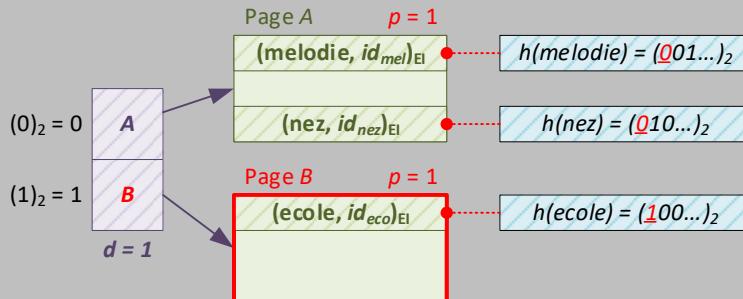
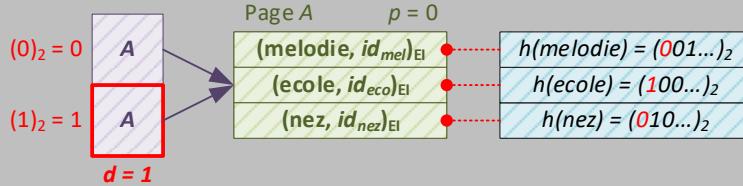
4. On ajoute dans la relation le mot `nez`, donc l'entrée d'index associée à la valeur `nez` est insérée dans l'index : d vaut 0, donc on « se dirige » automatiquement vers la seule case du répertoire de hachage dynamique qui « pointe » vers la page A. Celle-ci pouvant recevoir l'entrée d'index, on l'y insère.



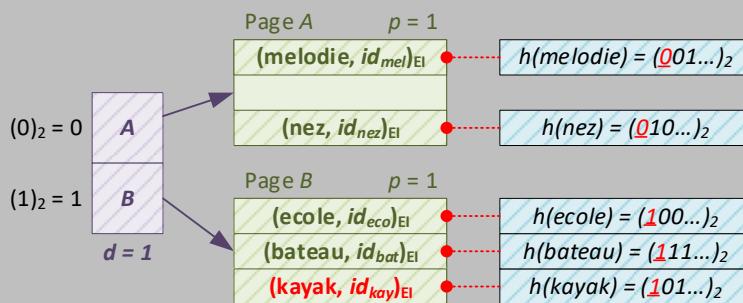
5. On ajoute dans la relation le mot `bateau`, donc l'entrée d'index de valeur `bateau` est insérée dans l'index : d vaut 0, donc on « se dirige » automatiquement vers la seule case du répertoire de hachage dynamique qui « pointe » vers la page A. Celle-ci n'a pas de place libre. On doit donc la diviser. Mais, pour cela, il faut que le répertoire de hachage dynamique soit assez grand, ce qui n'est pas le cas. On va donc dédoubler ce répertoire puis diviser la page A et enfin insérer l'entrée d'index (en prenant maintenant en compte le bit de poids fort de $h(v)$ pour sa pseudo-clé) dans la page appropriée.



Exemple (suite)

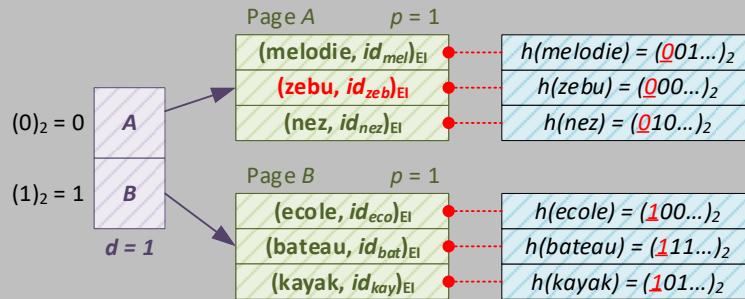


6. On ajoute dans la relation le mot kayak, donc l'entrée d'index de valeur kayak est insérée dans l'index : sa pseudo-clé (sur 1 bit, $d = 1$) vaut 1 ; la case d'indice 1 du répertoire de hachage dynamique « pointe » vers la page B qui contient de la place. On y insère donc l'entrée d'index.

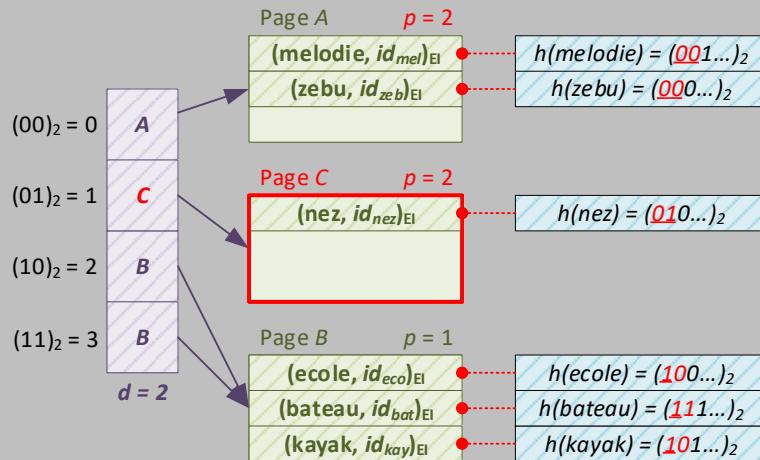
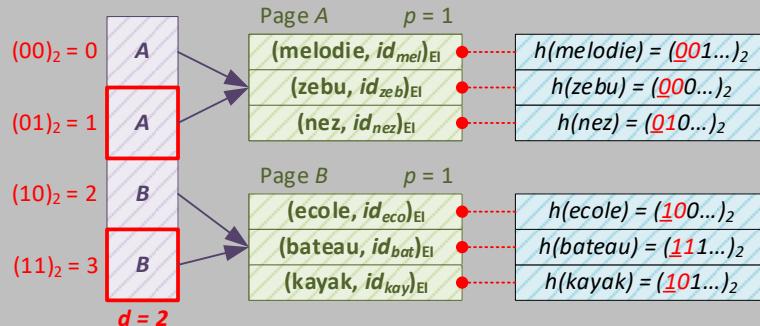


Exemple (suite)

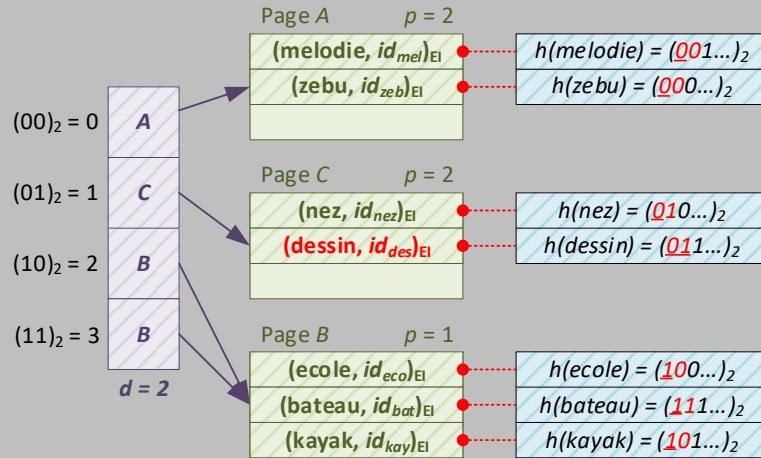
7. On ajoute dans la relation le mot zebu, donc l'entrée d'index de valeur zebu est insérée dans l'index : sa pseudo-clé (sur 1 bit, $d = 1$) vaut 0 ; la case d'indice 0 du répertoire de hachage dynamique « pointe » vers la page A qui contient de la place. On y insère donc l'entrée d'index.



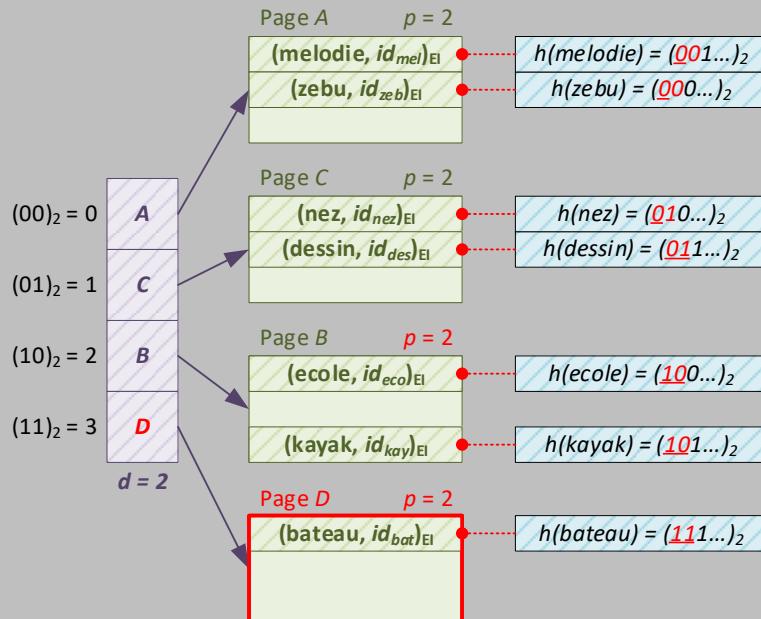
8. On ajoute dans la relation le mot dessin, donc l'entrée d'index de valeur dessin est insérée dans l'index : sa pseudo-clé (sur 1 bit, $d = 1$) vaut 0 ; la case d'indice 0 du répertoire de hachage dynamique « pointe » vers la page A qui ne contient plus de place. On doit donc la diviser. Mais, pour cela, il faut que le répertoire de hachage dynamique soit assez grand, ce qui n'est pas le cas. On va donc doubler ce répertoire puis diviser la page A et enfin insérer l'entrée d'index (en prenant maintenant en compte les 2 bits de poids fort de $h(v)$ pour sa pseudo-clé) dans la page appropriée.



Exemple (suite)

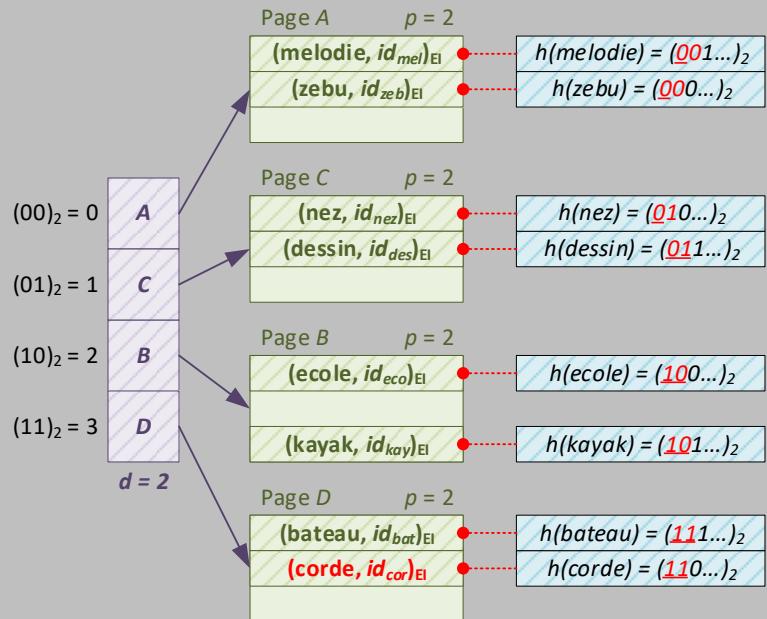


9. On ajoute dans la relation le mot *corde*, donc l'entrée d'index de valeur *corde* est insérée dans l'index : sa pseudo-clé (sur 2 bits, $d = 2$) vaut 3 ; la case d'indice 3 du répertoire de hachage dynamique « pointe » vers la page B qui ne contient plus de place. On doit donc la diviser (le répertoire de hachage dynamique est assez grand) puis insérer l'entrée d'index (en prenant toujours en compte les 2 bits de poids fort de $h(v)$ pour sa pseudo-clé) dans la page appropriée.





Exemple (fin)



8.2.2.3.3 Remarques et performances

La technique du hachage dynamique :

- Ne vise pas à optimiser la taille du répertoire de hachage dynamique : lorsque celui-ci est agrandi, il ne l'est pas du nombre de cases minimal nécessaire mais il est doublé. Ceci s'explique par le fait qu'il contient 2^d cases et, donc, quand d est incrémenté de 1, ce nombre est doublé.
- Vise à optimiser le nombre de pages pointées par les cases du répertoire de hachage dynamique : à tout moment, toutes les cases du répertoire de hachage dynamique pointent vers une et une seule page (il n'y a donc JAMAIS de cases « inutiles », i.e. ne pointant vers aucune page).
- Vise à optimiser le nombre de pages nécessaires au stockage des entrées d'index (quitte à faire pointer plusieurs cases du répertoire de hachage dynamique vers une même page).

Au niveau des performances, dans le cas le plus courant où le répertoire de hachage dynamique tient en mémoire de travail, la recherche se fait en 1 accès disque, quel que soit le nombre d'entrées d'index¹⁸⁷.



Remarque

On est sur un cas d'index quasi-optimal d'index mono-critère à accès par hachage :

- Les entrées d'index tiennent dans un nombre de pages très fréquemment proche de minimum requis,
- Les collisions sont très fréquemment bien réparties puisque :
 - Aucune case du répertoire n'est « perdue »,
 - Les listes chaînées de pages sont toutes de même longueur ET cette longueur est toujours égale à 1.
- Le coût d'usage de l'index est faible : le calcul des pseudo-clés est peu coûteux (manipulation de données binaires), seules les opérations de division/fusion de pages et de doublement/réduction du répertoire le sont « un peu ».

¹⁸⁷ Il est bien sûr tout à fait possible d'épingler dans la mémoire cache les pages qui le contiennent !

Les paramètres propres à un IMCAHD sont les suivants :

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$h(x)$ et N	-	Fonction de hachage telle que $0 \leq h(x) < N$
$nbMax_{entrees/page}^{fixe moy min max}(I(R, c))$	entrées d'index par page	Nombre (fixe ou moyen ou minimal ou maximal) d'entrées d'index que l'on peut stocker au plus dans une page d'un IMCAHD : $nbMax_{entrees/page}^{fixe moy min max}(I(R, c)) \stackrel{\text{rouge}}{=} \frac{T_{page}^{fixe}(*)}{T_{entree}^{fixe moy min max}(I(R, c))}$
$nb_{pagesEntrees}^{fixe moy min max}(I(R, c))$	pages	Nombre (fixe ou moyen ou minimal ou maximal) de pages contenant les entrées d'index : $nb_{pagesEntrees}^{fixe moy min max}(I(R, c)) \stackrel{\text{rouge}}{=} \frac{nb_{entrees}^{exact}(I(R, c))}{nbMax_{entrees/page}^{fixe moy min max}(I(R, c))}$
$T_{caseRepHD}^{fixe}(I(R, c))$	octets	Taille (forcément fixe) d'une case du répertoire de hachage dynamique : $T_{caseRepHD}^{fixe}(I(R, c)) = T_{idPage}^{fixe}(*)$
d	cases	Profondeur du répertoire : sert à déterminer le nombre de cases du répertoire de hachage dynamique ainsi que les pseudo-clés (d bits de poids fort des codes hachés).
$nb_{casesRepHD}^{exact}(I(R, c))$	octets	Nombre (forcément exact) de cases dans le répertoire de hachage dynamique : $nb_{casesRepHD}^{exact}(I(R, c)) = 2^d$
$nbMax_{casesRepHD/page}^{exact}(I(R, c))$	cases par page	Nombre maximal (forcément exact) de cases du répertoire de hachage dynamique au plus par page : $nbMax_{casesRepHD/page}^{exact}(I(R, c)) \stackrel{\text{rouge}}{=} \frac{T_{page}^{fixe}(*)}{T_{caseRepHD}^{fixe}(I(R, c))}$
$nb_{pagesRepHD}^{exact}(I(R, c))$	pages	Nombre (forcément exact) de pages occupées par le répertoire de hachage dynamique : $nb_{pagesRepHD}^{exact}(I(R, c)) = \frac{nb_{casesRepHD}^{exact}(I(R, c))}{nb_{casesRepHD/page}^{exact}(I(R, c))} \stackrel{\text{rouge}}{\rightarrow}$
$Coût_{usage}^{moy}(I(R, c))$	transferts de pages (en lecture)	Coût (moyen) d'usage de l'IMCAHD : $Coût_{usage}^{moy}(I(R, c)) = 1 \stackrel{\text{rouge}}{\underbrace{+1}_{\substack{\text{si repHD} \\ \text{non punaisé}}}}$

Tableau 49. Performances d'indexation : paramètres spécifiques aux IMCAHD

9 Optimisation de requêtes

SOMMAIRE DÉTAILLÉ DU CHAPITRE 9

9.1	Optimisation pré-évaluation : choix de la meilleure stratégie	252
9.1.1	Organisations d'opérateurs relationnels : les arbres de requête	254
9.1.1.1	Arbre de requête initial : les arbres jointures/sélection/projection	255
9.1.1.2	Transformations d'arbres de requêtes	257
9.1.1.2.1	Transformation T_1 : regroupement/éclatement/permuation des sélections	258
9.1.1.2.2	Transformation T_2 : permutation sélection \leftrightarrow projection	259
9.1.1.2.3	Transformation T_3 : permutation sélection \leftrightarrow jointure	261
9.1.1.2.4	Transformation T_4 : permutation projection \leftrightarrow jointure	261
9.1.1.2.5	Transformation T_5 : commutativité de la jointure	262
9.1.1.2.6	Transformation T_6 : associativité de la jointure	263
9.1.1.3	Traitemennt de l'explosion combinatoire : l'heuristique des ALGJ	263
9.1.1.3.1	Heuristique des arbres linéaires gauches de jointure (ALGJ)	265
9.1.1.3.2	Heuristique de descente des sélections puis des projections (DSP)	267
9.1.2	Coûts associés à un arbre de requête : les plans d'exécution	268
9.1.2.1	Notion de plan d'exécution	270
9.1.2.2	Coût d'un plan d'exécution	270
9.1.2.2.1	Coût d'une méthode de résolution d'un opérateur relationnel	272
9.1.2.2.2	Paramètres d'évaluation des coûts	273
9.1.2.2.2.1	Paramètres généraux (machine, OS, SGBD)	274
9.1.2.2.2.2	Paramètres d'une relation R	275
9.1.2.2.2.3	Paramètres d'un index I et propagation	277
9.1.2.2.2.4	Paramètres spécifiques à une requête	280
9.1.2.2.3	Autres notations utilisées	281
9.1.3	Coût de production des opérateurs relationnels	281
9.1.3.1	Coût de production d'une sélection	281
9.1.3.1.1	Évaluation d'une sélection au moyen d'un parcours séquentiel	282
9.1.3.1.2	Évaluation d'une équi-sélection au moyen d'un parcours indexé	283
9.1.3.2	Coût de production d'une jointure	284
9.1.3.2.1	Évaluation d'une jointure au moyen de boucles imbriquées	285
9.1.3.2.2	Évaluation d'une jointure au moyen de boucles optimisées	286
9.1.3.2.3	Évaluation d'une équi-jointure au moyen d'une boucle indexée	288
9.1.3.3	Coût de production d'une projection (sans élimination des doublons)	290
9.1.3.3.1	Évaluation d'une projection (sans élimination des doublons) au moyen d'un parcours séquentiel	290
9.1.3.3.2	Évaluation d'une projection (sans élimination des doublons) au moyen d'un parcours indexé	291
9.1.4	Coût d'écriture du résultat des opérateurs relationnels	293
9.1.4.1	Évaluation de la taille de la relation produite par une sélection	294
9.1.4.2	Évaluation de la taille de la relation produite par une jointure	295
9.1.4.3	Évaluation de la taille de la relation produite par une projection (sans élimination des doublons)	296
9.1.5	Étude de cas (partielle)	296
9.1.5.1	Paramètres donnés	297
9.1.5.2	Paramètres calculés	298
9.1.5.3	Arbre de requête initial : élaboration et étiquetage	300
9.1.5.4	Construction (partielle) d'arbres de requête équivalents	301
9.1.5.5	Étude (partielle) de coûts : étiquetage des relations et des opérateurs	303
9.1.5.5.1	Étiquetage des relations	303
9.1.5.5.2	Étiquetage des opérateurs relationnels	304
9.1.5.5.3	Étiquetage d'un arbre de requête	305
9.1.5.5.4	Exemple d'étiquetage d'arbres de requête équivalents	306
9.1.5.5.4.1	Étiquetage de l'arbre de requête initial A_{11}	306
9.1.5.5.4.2	Étiquetage des relations des arbres de requête A_{21}, A_{31} et A_{41}	310
9.1.5.5.4.3	Étiquetage des opérateurs relationnels des arbres de requête A_{21}, A_{31} et A_{41}	312
9.1.5.5.5	Conclusion de l'étude (partielle)	316
9.2	Prise en compte de l'optimisation pendant l'évaluation : les pipelines	321
9.2.1	Principe	321
9.2.2	Mise en œuvre et prise en compte par le moteur d'optimisation	322
9.2.3	Étude de cas (partielle)	324
9.2.3.1	Mise en œuvre d'une évaluation avec 1 pipeline	324
9.2.3.2	Mise en œuvre d'une évaluation avec 2 pipelines	327
9.2.3.3	Mise en œuvre avec 3 pipelines (ou plus)	327
9.2.3.4	Bilan	328
9.3	Paramètres liés à l'optimisation de requêtes	328

FIGURES DU CHAPITRE 9

Figure 137. Deux niveaux d'optimisation d'une requête.....	251
Figure 138. Optimisation <i>a priori</i>	253
Figure 139. Explosion combinatoire (requête \equiv arbre initial / arbres équivalents / plans d'exécution)	253
Figure 140. Exemple d'arbre de requête	254
Figure 141. Exemple d'arbre de requête initial	257
Figure 142. Transformation T_1 de regroupement/éclatement/permuation des sélections	258
Figure 143. Transformation T_2 de permutation sélection \leftrightarrow projection	259
Figure 144. Transformation T_3 de permutation sélection \leftrightarrow jointure	261
Figure 145. Transformation T_4 de permutation jointure \leftrightarrow projection	262
Figure 146. Transformation T_5 de commutativité de la jointure	262
Figure 147. Transformation T_6 d'associativité de la jointure	263
Figure 148. Forme générale d'un arbre linéaire gauche de jointure	265
Figure 149. Arbres linéaires gauches et droits de jointure	266
Figure 150. Arbre de requête résultant de l'application de l'heuristique DSP.....	268
Figure 151. Méthodes de résolution et plans d'exécution d'un arbre de requête	271
Figure 152. Répartition ordonnée des n-uplets quand R est indexée par un index groupé	278
Figure 153. Répartition équiprobable des n-uplets quand R est indexée par un index non-groupé	278
Figure 154. Optimisation (étude de cas) : arbre de requête initial A_{11}	301
Figure 155. Optimisation (étude de cas) : arbre de requête équivalent A_{12}	301
Figure 156. Optimisation (étude de cas) : arbres de requête équivalents A_{21} et A_{22}	302
Figure 157. Optimisation (étude de cas) : arbres de requête équivalents A_{31} et A_{32}	302
Figure 158. Optimisation (étude de cas) : arbres de requête équivalents A_{41} et A_{42}	303
Figure 159. Étiquetage d'une sélection	304
Figure 160. Étiquetage d'une jointure	305
Figure 161. Étiquetage d'une projection	305
Figure 162. Optimisation (étude de cas) : étiquetage des relations de l'arbre de requête A_{11}	308
Figure 163. Optimisation (étude de cas) : étiquetage complet de l'arbre de requête A_{11}	309
Figure 164. Optimisation (étude de cas) : étiquetage complet de l'arbre de requête A_{21}	315
Figure 165. Optimisation (étude de cas) : étiquetage complet de l'arbre de requête A_{31}	315
Figure 166. Optimisation (étude de cas) : étiquetage complet de l'arbre de requête A_{41}	316
Figure 167. Optimisation (étude de cas) : application de l'heuristique DSP à l'arbre de requête A_{41}	317
Figure 168. Transmission d'une relation temporaire sans pipeline	321
Figure 169. Transmission d'une relation temporaire avec pipeline	321
Figure 170. Niveaux dans l'arbre de requête A_{51} pour les évaluations en pipeline	323
Figure 171. Optimisation (étude de cas) : prise en compte d'un pipeline pour l'arbre A_{11}	324
Figure 172. Optimisation (étude de cas) : prise en compte d'un pipeline pour l'arbre A_{21}	325
Figure 173. Optimisation (étude de cas) : prise en compte d'un pipeline pour l'arbre A_{31}	325
Figure 174. Optimisation (étude de cas) : prise en compte d'un pipeline pour l'arbre A_{41} (cas 1)	326
Figure 175. Optimisation (étude de cas) : prise en compte d'un pipeline pour l'arbre A_{41} (cas 2)	326
Figure 176. Optimisation (étude de cas) : prise en compte de 2 pipelines pour l'arbre A_{41}	327

TABLEAUX DU CHAPITRE 9

Tableau 50. Exemples de nombres d'ordres linéaires de jointures sur n relations	264
Tableau 51. Comparaison des nombres d'ordres linéaires simples et gauches de jointures.....	266
Tableau 52. Plans d'exécution et coûts	271
Tableau 53. Paramètres généraux (machine, OS, SGBD)	274
Tableau 54. Paramètres relatifs aux constituants/attributs d'une relation R.....	275
Tableau 55. Paramètres relatifs aux n-uplets d'une relation R	276
Tableau 56. Paramètres relatifs à la globalité d'une relation R	277
Tableau 57. Paramètres relatifs à un index I	277
Tableau 58. Avantages et inconvénients de la propagation des index	279
Tableau 59. Paramètres spécifiques à une requête	280
Tableau 60. Notation et portion d'arbre de requête associée à une sélection	281
Tableau 61. Notation et portion d'arbre de requête associée à une jointure	285
Tableau 62. Notation et portion d'arbre de requête associée à une projection	290
Tableau 63. Bilan de la mise en œuvre d'1 ou 2 pipelines sur l'étude de cas	328
Tableau 64. Paramètres liés à l'optimisation de requêtes.....	328

ÉQUATIONS DU CHAPITRE 9

Équation 43. Nombre d'ordres linéaires de jointures sur n relations	263
Équation 44. Nombre d'ordres linéaires gauches de jointures sur n relations	265
Équation 45. Coût d'un plan d'exécution	271
Équation 46. Coût d'un opérateur relationnel (mis en œuvre par une méthode donnée)	273
Équation 47. Coût du chargement des n-uplets $c = v$ quand l'index $I(R, c)$ est groupé	278
Équation 48. Coût du chargement des n-uplets $c = v$ quand l'index $I(R, c)$ est non-groupé	278
Équation 49. Coût de production d'une sélection évaluée par un parcours séquentiel	282
Équation 50. Coût de production d'une sélection évaluée par un parcours indexé (I non-groupé)	284
Équation 51. Coût de production d'une sélection évaluée par un parcours indexé (I groupé)	284
Équation 52. Coût de production d'une jointure évaluée par des boucles imbriquées	286
Équation 53. Coût de production d'une jointure évaluée par des boucles optimisées	287
Équation 54. Coût de production d'une jointure évaluée par une boucle indexée (I non-groupé)	289
Équation 55. Coût de production d'une jointure évaluée par une boucle indexée (I groupé)	289
Équation 56. Coût de production d'une projection évaluée par un parcours séquentiel	291
Équation 57. Coût de production d'une projection évaluée par un parcours indexé	292
Équation 58. Coût d'écriture de la relation résultat d'un opérateur relationnel	293
Équation 59. Évaluation de la cardinalité de la relation résultat d'une sélection	294
Équation 60. Évaluation de la taille des données des n-uplets produits par une sélection	295
Équation 61. Évaluation de la cardinalité de la relation résultat d'une jointure	295
Équation 62. Évaluation de la taille des données des n-uplets produits par une jointure	296
Équation 63. Évaluation de la cardinalité de la relation résultat d'une projection	296
Équation 64. Évaluation de la taille des données des n-uplets produits par une projection	296
Équation 65. Triplet étiquetant chaque relation R d'un arbre de requête	303

L'accès à une BD est réalisé au travers d'un langage de requêtes (SQL ou un langage dérivé en relationnel). Les performances de l'évaluation des requêtes sont donc cruciales. Pour améliorer a priori ces performances d'évaluation, le SGBD peut mettre en place :

- *Un moteur d'optimisation* : celui-ci récupère la requête une fois analysée et « validée »¹⁸⁸ et cherche la manière la moins coûteuse de retourner la réponse à une requête. Rappelons qu'un langage de requêtes est déclaratif : l'utilisateur exprime sa requête mais pas la façon d'y répondre. Or, pour une requête donnée, il existe en général plusieurs stratégies pour construire la réponse. C'est au SGBD de choisir la stratégie optimale, *i.e.* la moins coûteuse, à mettre en œuvre pour calculer le résultat d'une requête. Ainsi, il est amené à élaborer différentes stratégies permettant de calculer ce résultat, à évaluer *a priori* le coût de ces différentes stratégies et enfin à déterminer laquelle est la stratégie la moins coûteuse : c'est celle-là qui sera effectivement mise en œuvre par le moteur d'évaluation pour calculer le résultat.



Remarque

Bien qu'il dépende de beaucoup de paramètres, nous exprimerons le coût *a priori* d'une stratégie de résolution d'une requête :

- Soit en nombre de transferts de pages,
- Soit en temps de transferts de pages (rappelons que le temps de lecture d'une page est rarement égal au temps d'écriture d'une page).

- *Une optimisation du moteur d'évaluation* : une fois la « meilleure » stratégie retenue par le moteur d'optimisation, elle est mise en œuvre par le moteur d'évaluation. Celui-ci peut être optimisé, notamment *via* la mise en place de mécanismes permettant de réduire le nombre de transferts de pages à effectuer pendant l'évaluation du résultat avec la stratégie choisie.

Le moteur d'optimisation intervient donc *a priori*, *i.e.* avant le calcul de la réponse à la requête. Puis, pendant ce calcul, le moteur d'évaluation peut lui-même mettre en œuvre d'autres mécanismes permettant d'optimiser le coût dudit calcul.

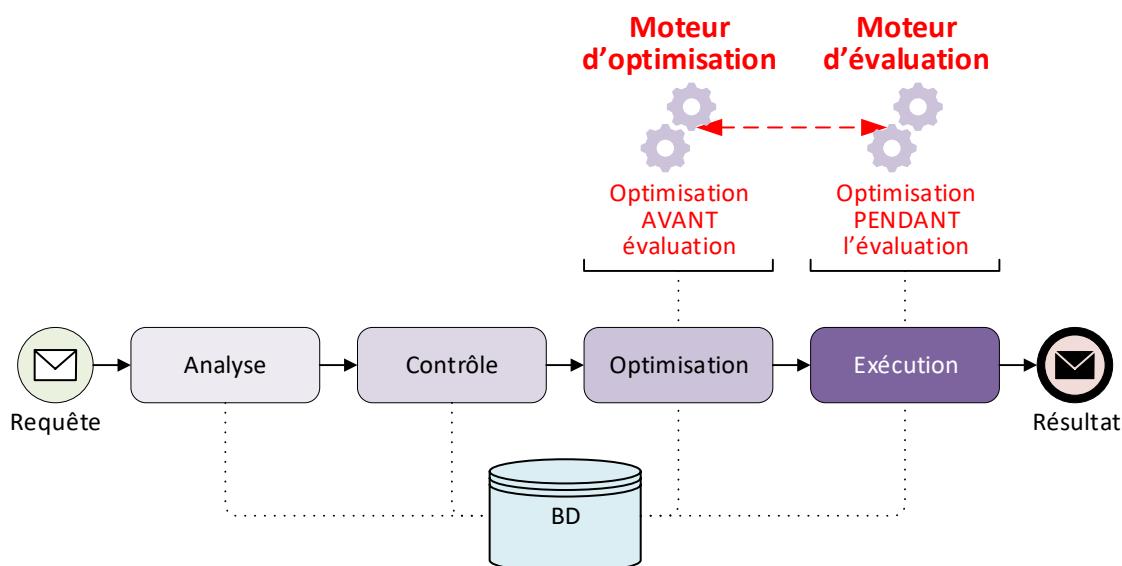


Figure 134. Deux niveaux d'optimisation d'une requête

¹⁸⁸ C'est-à-dire si les analyses lexicales, syntaxiques et sémantiques sont passées avec succès.



En pratique

Le moteur d'optimisation tient compte des possibilités d'optimisation offertes par le moteur d'évaluation pour trouver la meilleure stratégie de calcul de la réponse à une requête (d'où le « lien » entre les 2 moteurs dans la figure ci-dessus).

Nous nous cantonnerons dans ce chapitre à l'étude du moteur d'optimisation :

- Comment choisit-il la meilleure stratégie d'évaluation *a priori* ?
- Comment prend-il en compte pour cela les éventuelles possibilités d'optimisations offertes par le moteur d'évaluation ?

9.1 Optimisation pré-évaluation : choix de la meilleure stratégie

En SQL¹⁸⁹, les requêtes « masquent » en fait l'usage d'opérateurs de l'algèbre relationnelle.



Exemple

Considérons la requête SQL suivante :

```
SELECT l.Titre
      FROM Livre l, Editeur e
     WHERE l.Editeur = e.Nom
       AND e.Pays = "Espagne";
```

Derrière elle se cachent en réalité les opérateurs relationnels algébriques de projection, de jointure et de sélection.

Ainsi, grossièrement et même caricaturalement, l'évaluation d'une requête consiste en « une organisation des opérateurs relationnels algébriques » qui se cachent derrière elle ! L'optimisation de requête repose notamment sur l'idée suivante : **il existe plusieurs façons d'organiser puis d'évaluer des ensembles d'opérateurs de l'algèbre relationnelle tout en assurant de calculer la même réponse¹⁹⁰**. Chaque organisation identifie une ou plusieurs stratégies de calcul et a un coût de mise en œuvre *a priori* qui lui est propre. Donc, autant choisir l'organisation *a priori* la moins coûteuse et mettre en œuvre celle-là !

Pour réaliser ce travail, le moteur d'optimisation utilise les concepts d'**arbre de requête** et de **plan d'exécution** et opère comme suit :

1. Il traduit la requête en un **arbre de requête initial**.
2. Il construit tous les **arbres de requête équivalents** par application de **transformations d'arbres de requêtes**,
3. Pour chacun des arbres de requêtes obtenus (initial et équivalents), il cherche l'ensemble des **plans d'exécution** qu'on peut leur associer,
4. Il estime *a priori* le coût de mise en œuvre de chacun de ces plans d'exécution,
5. Il retient le plan d'exécution de coût *a priori* minimal.

¹⁸⁹ C'est aussi le cas sur les langages dérivés, bien sûr.

¹⁹⁰ Ou une réponse « similaire », i.e. dont seul l'ordre des attributs (des colonnes) change, les données rentrées (le contenu de la table) restant identiques.

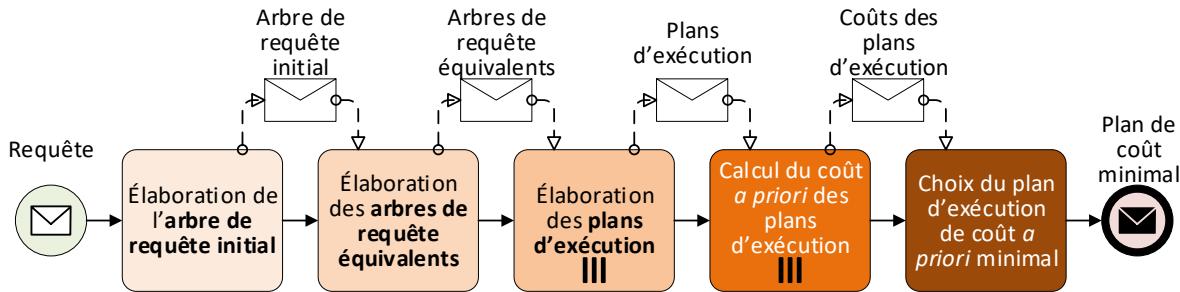


Figure 135. Optimisation *a priori*

Nous en reparlerons plus loin (cf. §9.1.1.3), mais le souci rapidement posé par cette méthode est celui de l’explosion combinatoire (cf. Figure 136) : assez vite, l’arbre de requête initial peut être associé à un grand nombre d’arbres de requêtes équivalents, eux-mêmes associés à un nombre vraiment grand de plans d’exécutions. Si on n’y prend pas garde, le coût pris par la recherche de l’optimum peut alors très souvent devenir plus important que le gain qui sera ensuite apporté. C’est pourquoi on développe, à côté de cette méthode, des heuristiques permettant de nettement réduire cette explosion combinatoire.

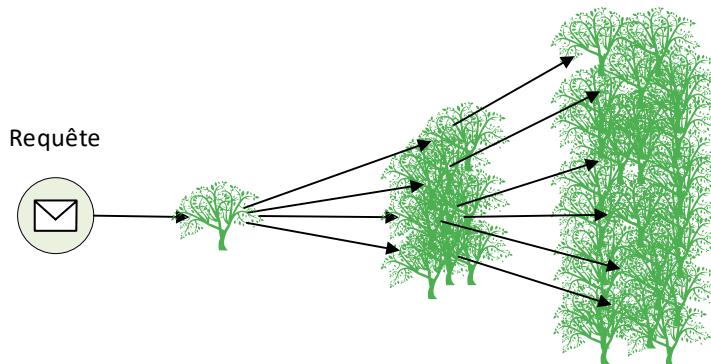


Figure 136. Explosion combinatoire (requête ≡ arbre initial / arbres équivalents / plans d'exécution)

Remarque

L’optimisation d’une requête SQL est un problème complexe, notamment si l’on prend en compte les opérateurs liés au tri et au groupement ainsi que les requêtes imbriquées !

Nous nous limiterons ici à l’étude de l’optimisation de requêtes SELECT ... FROM ... WHERE ... « simples » (*i.e.* sans opérations liées au tri ni au groupement) et sans imbrications. Nous utiliserons et étudierons donc uniquement les opérateurs relationnels algébriques suivants (seuls ceux-là se « cachant » derrière le type de requêtes auquel nous nous limitons) : les sélections (comparaisons faisant intervenir 1 seul attribut de leur unique relation d’entrée), les projections (faisant intervenir de 1 à n attributs de leur unique relation d’entrée) et les jointures (produits cartésiens avec comparaison entre 1 attribut de chacune de leurs deux relations d’entrées). Nous supposerons également qu’aucune valeur d’attribut n’est pas renseignée (*i.e.* est égale à NULL) au sein des relations manipulées par les requêtes. Enfin, nous autoriserons l’expression de conditions de sélection ou de jointure complexes mais seulement si elles n’utilisent que l’opérateur de conjonction AND (nous ne traiterons donc pas les disjonctions OR ni les négations NOT).

Ces 3 hypothèses simplificatrices sont valides pour tout le présent chapitre.



9.1.1 Organisations d'opérateurs relationnels : les arbres de requête

Comme nous venons de le dire, une même requête peut être associée à des « organisations » d'opérateurs relationnels algébriques. Ces « organisations » prennent la forme d'arbres de requête.



Définition : « arbre de requête »

Un **arbre de requête** est un arbre (*sic*) dans lequel :

- Les nœuds sont les opérateurs de l'algèbre relationnels sous-jacents à la requête,
- Les arcs sont des relations (réelles en bas de l'arbre, temporaires entre les opérateurs relationnels ou résultat en haut de l'arbre).

Un arbre de requête « s'interprète » de bas en haut (*i.e.* depuis les relations réelles sur lesquelles la requête est formulée, situées en bas, jusqu'au résultat, situé en haut).



Exemple (début)

Reprendons la requête SQL suivante :

```
SELECT l.Titre
  FROM Livre l, Editeur e
 WHERE l.Editeur = e.Nom
       AND e.Pays = "Espagne"
```

Elle peut par exemple¹⁹¹ être associée à l'arbre de requête suivant :

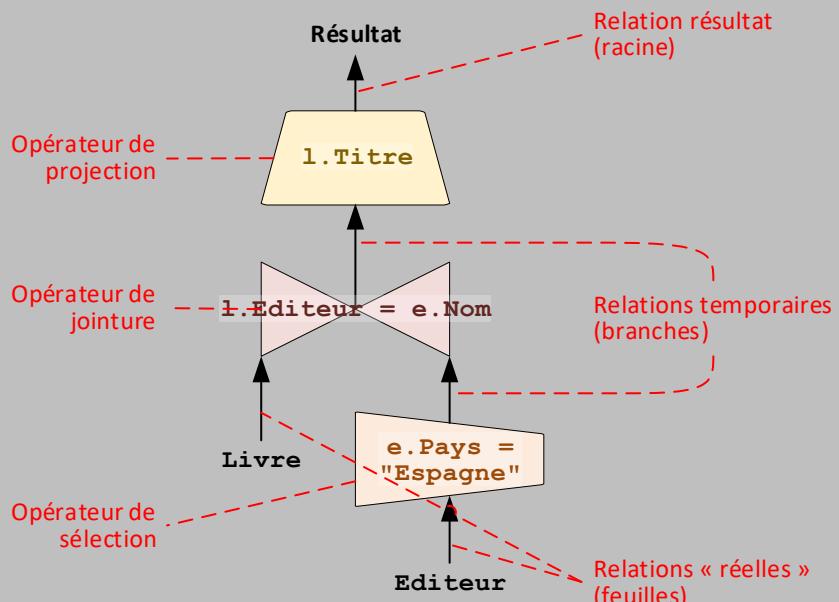


Figure 137. Exemple d'arbre de requête

¹⁹¹ Cet arbre n'est en fait qu'un des arbres de requête que l'on pourrait associer à cette requête.



Exemple (suite et fin)

Un tel arbre de requête « s'interprète » (et s'évalue) comme suit, de bas en haut :

- La relation `Editeur` est lue et passe dans une sélection qui ne retient en sortie que ses n-uplets pour lesquels l'attribut `Pays` vaut « Espagne » : les n-uplets retenus forment donc une relation temporaire R_1 (ayant le même modèle conceptuel que la relation `Editeur`)
- La relation `Livre` et la relation temporaire R_1 sont lues et passent dans une jointure qui ne retient en sortie que les concaténations de n-uplets pour lesquels la valeur de l'attribut `Editeur` (issu de la relation `Livre`) est égale à celle de l'attribut `Nom` (issu de la relation temporaire R_1 , donc indirectement de la relation `Editeur`) : les concaténations retenues sont les n-uplets d'une relation temporaire R_2 (dont le modèle conceptuel résulte de la concaténation de celui de la relation `Livre` et de la relation temporaire R_1 , donc indirectement de la relation `Editeur`).
- La relation temporaire R_2 est lue et passe dans une projection qui ne retient que l'attribut `Titre` (issu indirectement de la relation `Livre`) : les n-uplets produits forment le **résultat** de la requête (dont le modèle conceptuel est formé uniquement des attributs sur lesquels la projection a été réalisée).

9.1.1.1 Arbre de requête initial : les arbres jointures/sélection/projection

L'indéterminisme étant « l'ennemi » de l'informatique, on ne peut pas se permettre de choisir « au hasard » un premier arbre de requête pour ensuite construire les arbres qui lui sont équivalents : comment construire cet arbre initial ? Par rapport à quoi ?

Pour répondre à ce problème¹⁹², on met en place un ensemble de « règles » permettant de toujours construire, pour une requête donnée, un premier arbre de requête qui sera toujours le même pour ladite requête : on l'appelle **arbre de requête initial**.



Définition : « arbre de requête initial »

Un **arbre de requête initial** est le premier arbre de requête construit par un moteur d'optimisation pour une requête donnée. Sa construction est régie par un ensemble de règles et c'est à partir de lui que les arbres équivalents seront construits, par application des règles de transformation d'arbres de requête.

¹⁹² Qui peut paraître anecdotique mais qui ne l'est en fait pas du tout !

Très majoritairement, on adopte les règles de construction suivantes pour cet arbre de requête initial :

- Il a la forme d'un arbre **jointures/sélection/projection**,

Définition : « arbre de requête jointures/sélection/projection »

Un **arbre de requête jointures/sélection/projection** est un arbre de requête (*sic*) dans lequel :

- Toutes les jointures (s'il y en a dans la requête) sont faites en bas de l'arbre,
- S'il y a dans la requête une ou plusieurs sélections, un seul opérateur de sélection apparaît dans l'arbre, immédiatement au-dessus des éventuelles jointures (si la requête contient plusieurs conditions de sélection, la condition de la sélection est une conjonction de toutes ces conditions),
- S'il y a dans la requête une projection, un seul opérateur de projection apparaît au sommet de l'arbre.

Un tel arbre de requête contient donc, de bas en haut :

- 0 à n jointures (d'où le s à *jointures* dans le nom de ces arbres),
- 0 ou 1 sélection (d'où l'absence de s à *sélection* dans le nom de ces arbres),
- 0 ou 1 projection (d'où l'absence de s à *projection* dans le nom de ces arbres).

- Les relations sont introduites de gauche à droite, dans la mesure du possible (*i.e.* en respectant **prioritairement**, s'il y en a, les jointures à faire telles qu'elles sont écrites dans la requête), selon un ordonnancement prédéfini (par ordre dans lequel elles apparaissent dans la clause `FROM` de la requête ou alors par ordre croissant ou décroissant de cardinalité, ...).

Exemple (début)

Soit la requête suivante :

```
SELECT l.Titre
  FROM Livre l, Editeur e, Auteur a
 WHERE l.Editeur = e.Nom
       AND l.Auteur = a.Nom
       AND a.Pays = "France"
       AND e.Pays = "Espagne"
       AND l.Annee >= 2000
```

Cette requête fait apparaître :

- 2 jointures : entre les relations `Livre` et `Editeur` avec pour condition de jointure `l.Editeur = e.Nom` et entre les relations `Livre` et `Auteur` avec pour condition de jointure `l.Auteur = a.Nom`,
- 3 sélections : sur la relation `Auteur` avec pour condition de sélection `a.Pays = "France"` et sur la relation `Editeur` avec pour condition de sélection `e.Pays = "Espagne"` et encore sur la relation `Livre` avec pour condition de sélection `l.Annee >= 2000`.
- 1 projection sur l'attribut `Titre` de la relation `Livre`.

Exemple (suite et fin)

En supposant que, pour construire l'arbre de requête initial, on introduit les relations par ordre croissant de leur cardinalité et, par ailleurs, que l'on a $\text{card}(\text{Livre}) > \text{card}(\text{Auteur}) > \text{card}(\text{Editeur})$, alors, l'arbre de requête initial, pour cette requête, est toujours le suivant :

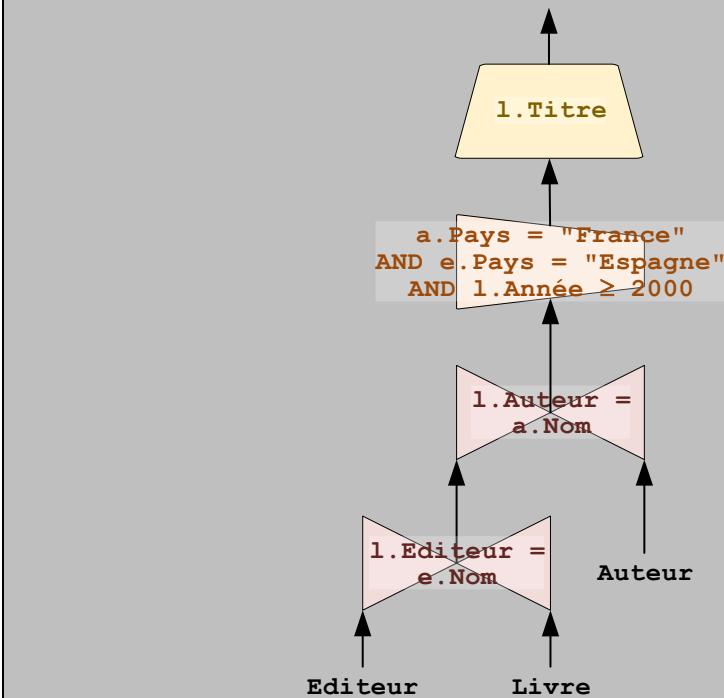


Figure 138. Exemple d'arbre de requête initial

Les relations sont introduites par ordre croissant de leur cardinalité, on introduit donc en premier la relation `Editeur` (celle de plus petite cardinalité). Logiquement, en suivant l'ordre croissant des cardinalités, on devrait introduire ensuite la relation `Auteur` mais il n'existe aucune jointure entre les relations `Editeur` et `Auteur`. En revanche, la relation `Editeur` est jointe avec la relation `Livre` (et uniquement avec elle) : c'est donc cette relation `Livre` qui est introduite en second. Enfin, la relation `Livre` participant à une jointure avec la relation `Auteur`, cette dernière est enfin introduite.

9.1.1.2 Transformations d'arbres de requêtes

On l'a déjà évoqué, il existe plusieurs façons d'organiser les opérateurs relationnels algébriques « se cachant » derrière une requête tout en conservant la même réponse (heureusement ! ☺). Autrement dit, plusieurs arbres de requêtes peuvent être associés à une même requête : chacun d'entre eux aura un ou des coûts d'évaluation différents¹⁹³. Pour trouver ces arbres de requête équivalents, on met en œuvre des transformations : elles ont pour but de créer de nouveaux arbres de requête équivalents.

¹⁹³ Mais nous reparlerons des coûts plus tard...



Définition : « arbres de requête équivalents »

Des **arbres de requête** sont **équivalents** s'ils sont différents (*i.e.* si les opérateurs relationnels algébriques apparaissant en leur sein sont différents et/ou organisés différemment) mais fournissent en résultat une relation similaire.



Définition : « relations similaires »

Deux **relations** sont **similaires** si elles contiennent globalement les mêmes données et qu'elles diffèrent uniquement par l'ordre d'apparition des attributs dans leur modèle conceptuel (le nombre de ces attributs ainsi que le nom, le type et la sémantique de chacun d'entre eux et les valeurs de ces attributs pour chaque n-uplet doivent en revanche être strictement identiques entre relations similaires).



Exemple

Soit la relation R_1 et la relation R_2 dont les modèles conceptuels sont les suivants :

$$\begin{aligned} R_1 &(\text{Prénom}, \text{Nom}, \text{Âge}) \\ R_2 &(\text{Nom}, \text{Prénom}, \text{Âge}) \end{aligned}$$

Seul l'ordre d'apparition des attributs distingue ces 2 modèles conceptuels. Donc, si elles contiennent les mêmes données (*i.e.* que les n-uplets qu'elles contiennent sont les mêmes à l'ordre des valeurs d'attributs près), alors elles sont similaires.

Ces transformations sont au nombre de 6 et sont toutes réversibles...

9.1.1.2.1 Transformation T_1 : regroupement/éclatement/permutation des sélections

Une requête peut contenir une ou plusieurs sélections. Celles-ci peuvent sans souci être regroupées si elles se succèdent ou, au contraire, une sélection dont la condition est une conjonction de conditions plus simples peut être éclatée en sélections successives, sans que cela ne modifie le résultat¹⁹⁴.

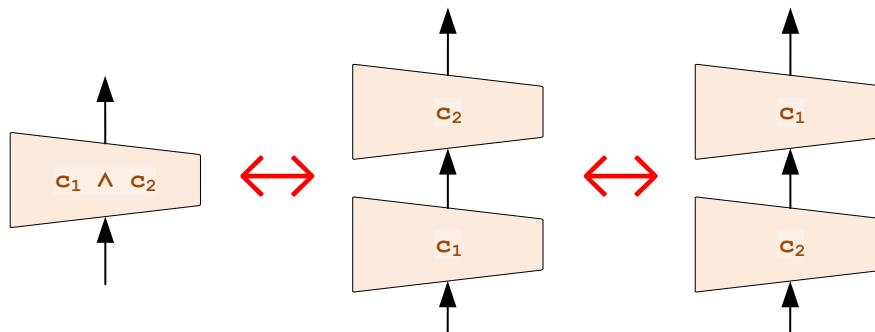


Figure 139. Transformation T_1 de regroupement/éclatement/permutation des sélections

¹⁹⁴ En revanche, cela aura une incidence sur le coût, nous le verrons plus loin...

Exemple

Soit la requête suivante :

```
SELECT *
FROM Editeur e
WHERE e.Pays = "Espagne"
    AND e.Nom LIKE "%libro"
```



Dans cette requête, on a 2 sélections : l'une porte sur l'attribut Pays et l'autre sur l'attribut Nom de la relation Editeur. La réponse sera toujours la même selon que l'on récupère les n-uplets qui nous intéressent :

- En ne « conservant » que ceux qui vérifient les 2 conditions d'un coup,
- En ne « conservant » que ceux qui vérifient la condition sur l'attribut Pays puis, parmi ceux-là et dans un second temps, ceux qui vérifient la condition sur l'attribut Nom,
- En ne « conservant » que ceux qui vérifient la condition sur l'attribut Nom puis, parmi ceux-là et dans un second temps, ceux qui vérifient la condition sur l'attribut Pays.

Dans les 3 cas, le résultat est même plus que similaire : il est strictement identique !

9.1.1.2.2 Transformation T_2 : permutation sélection \leftrightarrow projection

Il est possible de permutez sélection et projection, c'est-à-dire de faire passer après (*i.e.* au-dessus) une projection une sélection qui était initialement faite immédiatement avant (*i.e.* sous) cette projection (et vice-versa).



Attention

La transformation n'est pas aussi simple que cela : **dans de nombreux cas, le résultat obtenu après transformation ne serait ni identique ni même similaire ni même calculable !!!** Afin de s'assurer de pouvoir calculer un résultat similaire ou identique, il faut « compléter » la simple permutation...

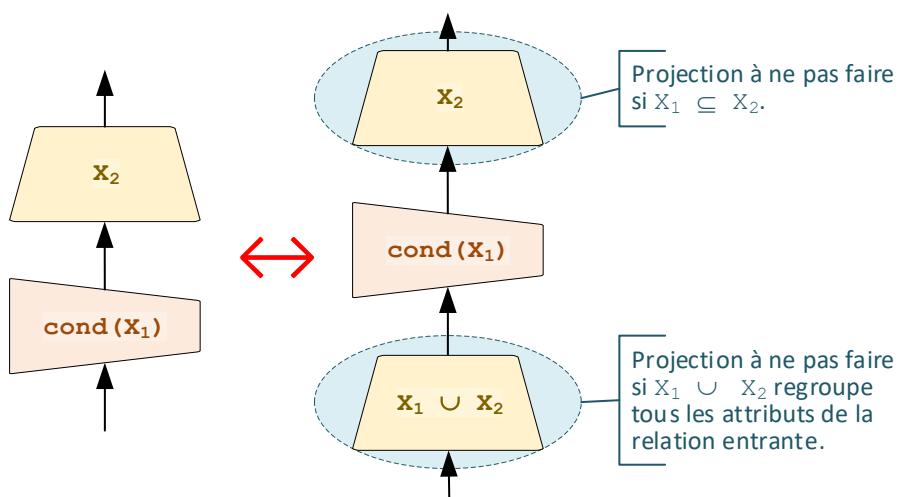


Figure 140. Transformation T_2 de permutation sélection \leftrightarrow projection

Question

Quels sont les problèmes pouvant survenir impliquant que la transformation ne puisse pas se faire par une simple permutation des 2 opérateurs ?

Réponse

Ils sont principalement deux (nous les expliquons ici au travers de la transformation vue de gauche à droite mais des problèmes similaires peuvent être causés par la transformation vue de droite à gauche) :

1. Si on fait passer la projection sur le constituant X_2 sous la sélection dont la condition porte sur le constituant X_1 , la projection est donc faite avant et, par définition, ne conserve que les attributs faisant partie du constituant X_2 . Rien ne permet d'assurer, donc, qu'il reste, à l'issue de la projection, tous les attributs faisant partie du constituant X_1 sur lequel porte la sélection. Il est donc tout à fait plausible que la sélection ne puisse même pas être calculée lorsqu'elle est réalisée après la projection ! Après transformation, avant la sélection, il est donc nécessaire de projeter sur le constituant $X_1 \cup X_2$ afin d'être sûr d'avoir au moins tous les attributs sur lesquels porte la sélection et, donc, être certain de pouvoir la calculer... Fort logiquement, si le constituant $X_1 \cup X_2$ contient tous les attributs de la relation d'entrée, on ne fait pas la projection initiale du tout (cela aurait un coût pour finalement « ne rien faire » !).
2. Même si on fait ce qui est indiqué ci-dessus (*i.e.* la projection sur le constituant $X_1 \cup X_2$ sous la sélection dont la condition porte sur le constituant X_1), le résultat obtenu n'est toujours pas similaire ! Avant transformation (à gauche), le résultat ne contient que les attributs du constituant X_2 sur lequel la projection a été faite. Si on fait, après transformation, une projection sur le constituant $X_1 \cup X_2$ puis une sélection dont la condition porte sur le constituant X_1 , alors le résultat contient les attributs du constituant $X_1 \cup X_2$. Les 2 résultats ne sont donc pas similaires. Il convient donc, après avoir fait passer la projection sous la sélection, de remettre une projection sur le constituant X_2 pour assurer la similarité des résultats... Fort logiquement si $X_1 \subseteq X_2$ (et, donc, si $X_1 \cup X_2 = X_2$), on ne fait pas la projection finale du tout (cela aurait un coût pour, là encore, finalement « ne rien faire »).

Question (subsidiaire)

Mais, du coup, quel est l'intérêt de réaliser la transformation de gauche à droite : il y a plus d'opérateurs dans l'arbre après transformation que dans l'arbre avant !

Réponse

Cela est vrai : l'arbre de droite contient plus d'opérateurs (3) que l'arbre de gauche (2). Cela n'implique cependant pas qu'il sera plus coûteux à évaluer : il y aura bien des cas où son évaluation coûtera même *a priori* moins cher !



9.1.1.2.3 Transformation T_3 : permutation sélection \leftrightarrow jointure

Il est aussi possible de permuter une sélection et une jointure (plus « facilement » qu'une sélection et une projection !).

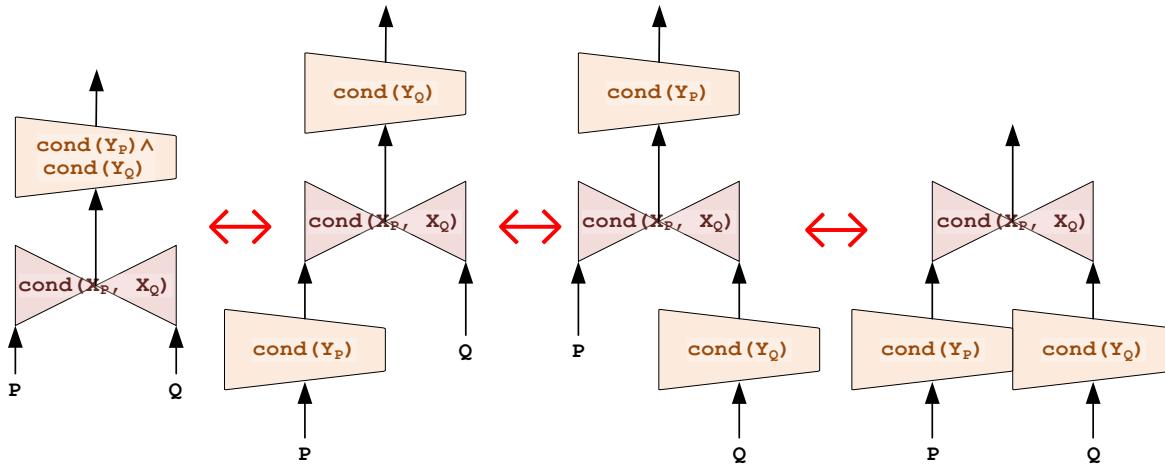


Figure 141. Transformation T_3 de permutation sélection \leftrightarrow jointure

Exemple

Considérons la requête SQL suivante :

```
SELECT l.Titre
FROM Livre l, Editeur e
WHERE l.Editeur = e.Nom
    AND e.Pays = "Espagne"
    AND l.Année ≥ 2000
```



Que l'on conserve les n-uplets « liés » à des éditeurs espagnols avant ou après la jointure ne change rigoureusement rien au résultat. De même, que l'on conserve les n-uplets « liés » à des livres parus en 2 000 ou après ne change là encore rigoureusement rien au résultat.



Remarque

Dans le cas de l'arbre de gauche, la sélection, si elle est composée, peut bien sûr être décomposée par la transformation T_1 de regroupement/éclatement/permutation des sélections (cf. §9.1.1.2.1).

9.1.1.2.4 Transformation T_4 : permutation projection \leftrightarrow jointure

Il est possible de permuter jointure et projection, c'est-à-dire de faire passer après (i.e. au-dessus) une projection une jointure qui était initialement faite immédiatement avant (i.e. sous) cette projection (et vice-versa).



Attention

Là encore (cf. §9.1.1.2.2), la transformation n'est pas aussi simple que cela : **dans de nombreux cas, le résultat obtenu après transformation ne serait ni identique ni même similaire ni même calculable !!!** Afin de s'assurer de pouvoir calculer un résultat similaire ou identique, il faut « compléter » la simple permutation...

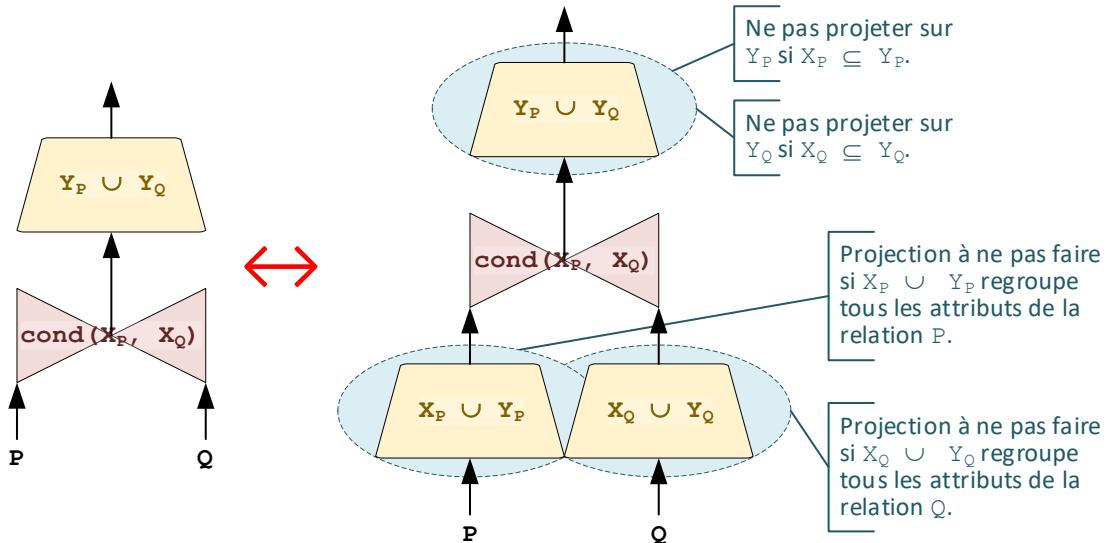


Figure 142. Transformation T_4 de permutation jointure ↔ projection



Remarque

Les problèmes pouvant être posés par une simple permutation, ainsi que leur résolution, sont similaires à ceux qui se posent dans le cas de la transformation T_2 de permutation sélection ↔ projection, ainsi que pour leur résolution (cf. §9.1.1.2.2).

9.1.1.2.5 Transformation T_5 : commutativité de la jointure

Une jointure peut être vue comme l'application d'une sélection sur le résultat d'un produit cartésien. De façon similaire à celle de la multiplication (« produit ») en mathématiques, le produit cartésien, donc la jointure, est un opérateur commutatif : le résultat de la jointure des relations P et Q est similaire au résultat de la jointure des relations Q et P ! 😊



Question

Pourquoi similaire et pas identique ? Après tout, pour reprendre l'analogie avec la multiplication, $2 \times 3 = 3 \times 2$!

Réponse

Parce que l'ordre des colonnes n'est pas le même : le résultat de la jointure des relations P et Q contient d'abord les colonnes issues de la relation P puis celles issues de la relation Q. C'est l'inverse pour le résultat de la jointure des relations Q et P.

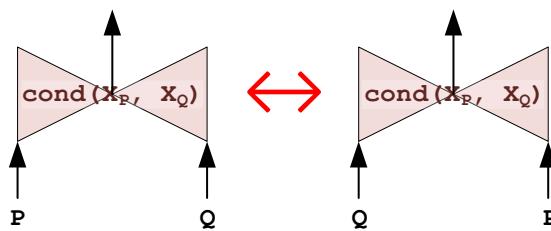


Figure 143. Transformation T_5 de commutativité de la jointure

9.1.1.2.6 Transformation T_6 : associativité de la jointure

Enfin, toujours par analogie entre la jointure et la multiplication, l'opérateur de jointure est associatif ! Donc, joindre 3 relations P , Q et R peut se faire de 2 façons différentes en gardant des résultats rigoureusement identiques :

- Joindre les relations P et Q puis joindre le résultat avec la relation R (i.e. $(P \times Q) \times R$),
- Joindre la relation P avec le résultat de la jointure entre les relations Q et R (i.e. $P \times (Q \times R)$).

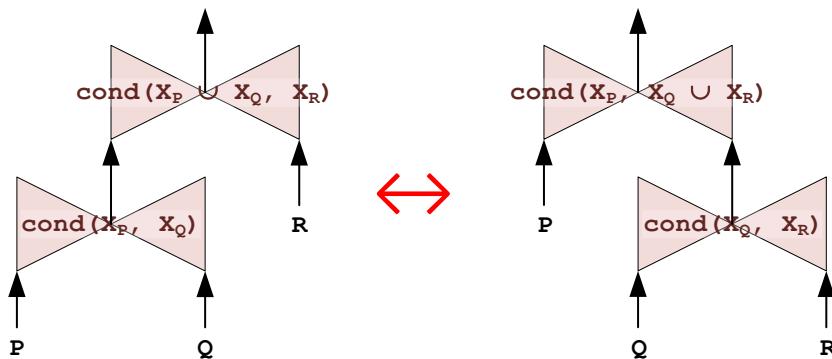


Figure 144. Transformation T_6 d'associativité de la jointure



Remarque

Ici, les résultats sont bien identiques puisque l'ordre des colonnes est préservé.

9.1.1.3 Traitement de l'explosion combinatoire : l'heuristique des ALGJ

Nous venons de le voir, l'optimisation débute par la construction d'un arbre de requête initiale et par construction des arbres de requêtes équivalents par application des transformations d'arbres. Cette méthode présente cependant un inconvénient majeur : plus le nombre de relations intervenant dans la requête est grand, plus le nombre d'arbres de requêtes équivalents à construire augmente. Par exemple, simplement en considérant le nombre d'ordres de jointures que l'on peut construire sur n relations, il est démontré qu'il vaut :

$$Nb_{ordJoin}(n \text{ relations}) = \frac{(2 \times (n - 1))!}{(n - 1)!}$$

Équation 43. Nombre d'ordres linéaires de jointures sur n relations



Définition : « ordre linéaire de jointure »

On appelle **ordre linéaire de jointure** une façon particulière (parmi tant d'autres) de réaliser les jointures entre n relations.



Remarque

Les différents ordres de jointure d'un même ensemble de n relations, peuvent s'obtenir en appliquant sur un ordre de jointure initial les transformations d'arbres T_5 (cf. §9.1.1.2.5) et T_6 (cf. §9.1.1.2.6) liées à la commutativité et l'associativité de la jointure.

Ce nombre croît donc très vite quand n augmente :

Nombre n de relations à joindre	Nombre d'ordres linéaires de jointures sur ces n relations
3	12
5	1 680
7	665 280
...	...

Tableau 50. Exemples de nombres d'ordres linéaires de jointures sur n relations

On sent donc la multiplication (exponentielles) du nombre d'arbres de requêtes équivalents à construire pour n relations. Sans compter que nous n'avons parlé ici que du nombre d'ordres linéaires de jointures, donc uniquement du nombre de façons de réaliser les jointures entre ces n relations, donc uniquement en considérant les transformations T_5 (cf. §9.1.1.2.5) et T_6 (cf. §9.1.1.2.6). N'oubliez pas que les arbres de requêtes équivalents se construisent aussi en appliquant les transformations T_1 (cf. §9.1.1.2.1), T_2 (cf. §9.1.1.2.2), T_3 (cf. §9.1.1.2.3) et T_4 (cf. §9.1.1.2.4) et imaginez alors le nombre d'arbres de requêtes équivalents que l'on peut obtenir sur n relations !!!

On se rend donc facilement compte que, si on cherche à construire tous les arbres de requête équivalents à l'arbre de requête initial, ce travail sera tellement coûteux qu'il est fort probable que le temps perdu durant « l'optimisation » pour construire tous ces arbres équivalents sera nettement plus grand que le temps éventuellement gagné lors de l'évaluation de la requête (d'où les guillemets rajoutés autour « d'optimisation ») ! 😞 Pour pallier ce problème, on peut adopter des heuristiques : celles-ci visent à limiter le nombre d'arbres équivalents que l'on va chercher à construire tout en garantissant d'optimiser « convenablement ».



Remarque

Il est alors possible que l'on « passe à côté » de l'optimalité mais, statistiquement on va tout de même s'en approcher. Le coût économisé (en ne construisant pas tous les arbres de requête équivalents) sera compensé par la faible différence entre l'optimum que l'on aurait pu atteindre et l'approchant que l'on trouvera.

Encore une fois, tout est une question de compromis...

Nous présentons ci-dessous 2 des principales heuristiques utilisées. Notez qu'elles sont compatibles l'une avec l'autre.

9.1.1.3.1 Heuristique des arbres linéaires gauches de jointure (ALGJ)

Nous avons vu ci-dessus que le nombre d'ordres de jointure croissait très fortement avec le nombre de relations à joindre. Pour minimiser le nombre d'ordres de jointure à considérer, on se limite en général aux arbres de requêtes dits **arbres linéaires gauches de jointure** (*left-deep join trees*).

Définition : « arbre linéaire gauche de jointure (ALGJ) »

Un **arbre linéaire gauche de jointure (ALGJ)** est un arbre de requête dans lequel une jointure ne peut jamais intervenir (ni directement ni indirectement) sous la branche droite d'un autre opérateur de jointure situé plus haut dans l'arbre de requête.

Ces ALGJ ont la « forme générale » suivante (les sous-arbres droits des opérateurs de jointure ne contiennent aucun opérateur de jointure) :

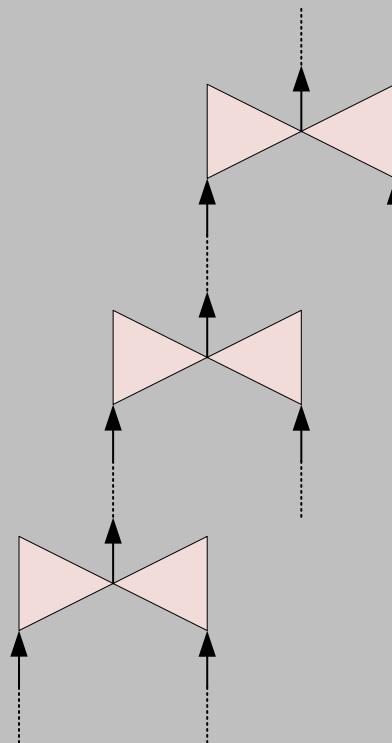


Figure 145. Forme générale d'un arbre linéaire gauche de jointure

Se cantonner à ne construire que des arbres de requête équivalents qui sont des arbres linéaires gauches de jointure réduit drastiquement le nombre d'arbres à construire. En raisonnant encore une fois sur les différentes façons de réaliser des jointures, on s'en rend compte déjà aisément (on parle alors d'**ordres linéaires gauches de jointure**) : il est démontré que, pour n relations, il y a un nombre d'ordres linéaires gauches de jointures qui est égal à...

$$Nb_{ordJoinLeft}(n \text{ relations}) = n!$$

Équation 44. Nombre d'ordres linéaires gauches de jointures sur n relations

On peut donc comparer les ordres de grandeur...

Nombre n de relations à joindre	Nombre d'ordres linéaires de jointures sur ces n relations	Nombres d'ordres linéaires <u>gauches</u> de jointures sur ces n relations
3	12	6
5	1 680	120
7	665 280	5 040
...

Tableau 51. Comparaison des nombres d'ordres linéaires simples et gauches de jointures

Remarques

L'heuristique des ALGJ est particulièrement intéressante lorsqu'il existe un index (cf. §8) sur l'attribut de jointure ou en cas de résolution pipeline (cf. §9.2). De plus, il est préférable de faire disparaître au plus tôt les produits cartésiens au profit des jointures, les produits cartésiens ayant un coût très élevé. Enfin, il existe aussi une heuristique consistant à ne construire que des arbres linéaires droits de jointure (ALDJ) :

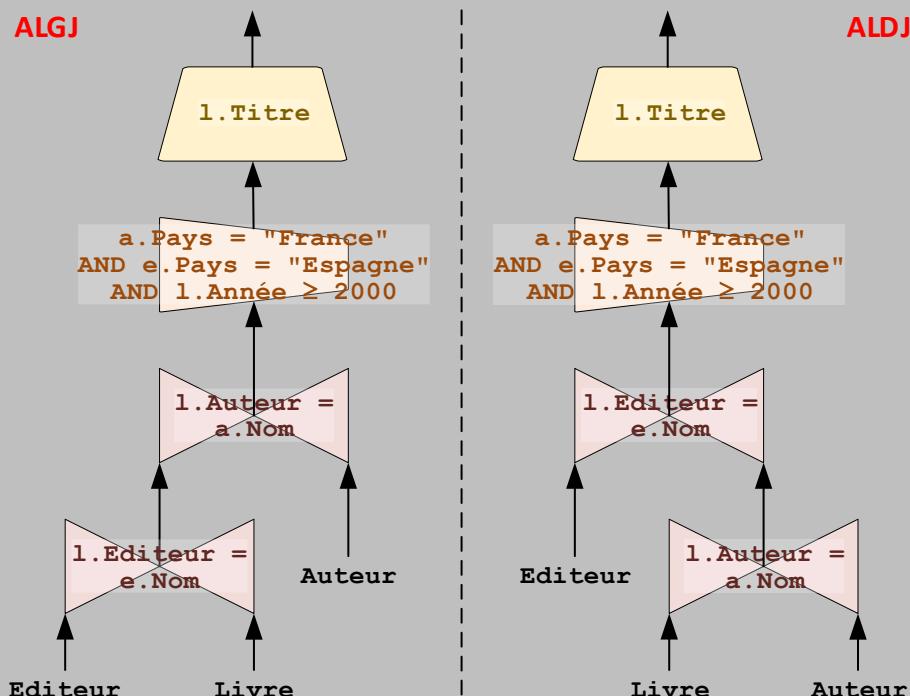


Figure 146. Arbres linéaires gauches et droits de jointure

Cependant, les 2 heuristiques (*i.e.* ne construire que des ALGJ ou que des ALDJ) ne sont pas équivalentes : nous le verrons (cf. §9.1.3.2), les 2 branches qui rentrent dans une jointure (par la gauche et par la droite) ne « jouent pas le même rôle » dans la jointure et n'ont pas la même incidence sur le coût *a priori* de la jointure. Parmi ces 2 heuristiques, la plus rencontrée, et de loin, est bien celle qui consiste à ne construire que des ALGJ...

9.1.1.3.2 Heuristique de descente des sélections **puis** des projections (DSP)

L'objectif de cette méthode d'optimisation est de minimiser la cardinalité des relations temporaires générées pendant la résolution d'une requête (*i.e.* des relations transmises d'un opérateur relationnel de l'arbre de requête au suivant, situé immédiatement au-dessus). L'idée est de partir d'un arbre de requête initial « classique » (*i.e.* de la forme jointures/sélection/projection) et d'arriver à un ensemble d'arbres de requêtes de la forme sélections/projections/jointures (dans le cas idéal). Ainsi, en réalisant au plus tôt les sélections **puis** les projections, on espère minimiser au maximum les paramètres des relations temporaires et, donc, le coût des opérateurs relationnels situés plus haut dans le plan d'exécution retenu (notamment des jointures, donc).

Définition : « arbre de requête sélections/projections/jointures »

Un arbre de requête sélections/projections/jointures est un arbre de requête (*sic*) dans lequel :

- Les sélections sont faites le plus tôt possible (elles apparaissent donc le plus bas possible dans l'arbre de requête),
- Les projections sont faites immédiatement après les sélections (donc « juste » au-dessus),
- Les jointures sont faites le plus tard possible (elles apparaissent donc le plus haut possible dans l'arbre de requête).



Un tel arbre de requête contient donc, sur chaque branche de l'arbre vue de bas en haut :

- 0 à n sélections,
- 0 ou n projections,
- 0 ou n jointures (**éventuellement intercalées avec des projections en raison des transformations d'arbres de requête T_2 et T_4 .**)

Les arbres de requête sont ainsi générés de la façon suivante :

1. La requête est vue au travers d'un arbre de requête initial « classique » de la forme jointures/sélection/projection,
2. Si elle existe, la sélection de l'arbre de requête initial est décomposée en sélections dont les conditions sont plus simples en appliquant la transformation T_1 (l'idée est de ne conserver dans une sélection que les conditions de sélection qui ont trait à des attributs d'une même relation),
3. Les sélections ainsi obtenues sont descendues le plus bas possible sur leur branche respective (*i.e.* celle qui correspond à la relation possédant les attributs sur lesquels leur condition porte) en appliquant la transformation T_3 ,
4. La projection est descendue sous les jointures (mais reste au-dessus des sélections descendues à l'étape précédente) à l'aide de la transformation T_4 ,
5. Les éventuelles projections inutiles (*i.e.* qui portent sur tous les attributs de la relation à laquelle elles s'appliquent) sont simplement éliminées : l'arbre obtenu est bien de la forme sélections/projections/jointures,
6. Des arbres de requête équivalents sont générés par application des transformations T_5 et T_6 .



Remarque

Notamment en raison de l'application de la transformation T_4 (permutation projection \leftrightarrow jointure), le nombre d'opérateurs relationnels présents dans l'arbre obtenu à l'issue de l'étape 5 est très probablement plus grand que celui associé à l'arbre de requête initial. Cependant, les relations à l'entrée des jointures étant réduites, le coût global devrait être bien moindre (on l'espère !)...



Exemple

L'arbre de requête ci-dessous résulte de l'application de l'heuristique de descente des sélections puis des projections (DSP) appliquée à l'arbre linéaire gauche de jointure de la Figure 146 :

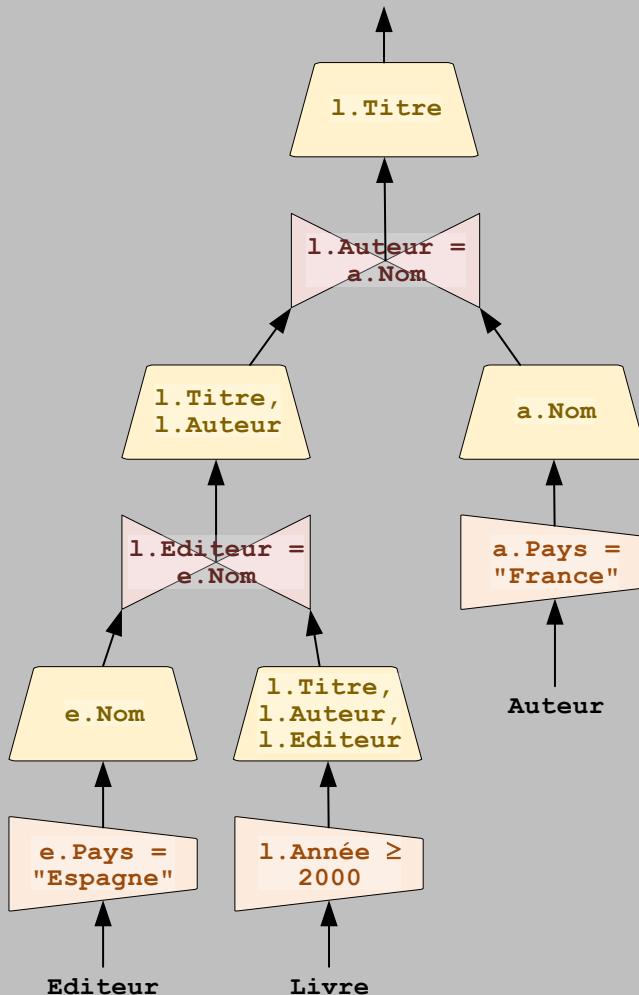


Figure 147. Arbre de requête résultant de l'application de l'heuristique DSP

9.1.2 Coûts associés à un arbre de requête : les plans d'exécution

Chaque arbre de requête est donc une organisation particulière d'un ensemble d'opérateurs relationnels. Il se trouve que, pour chacun de ces opérateurs, il existe plusieurs façons de « le calculer » : on parle de **méthode de résolution**, les différentes méthodes de résolution applicables à un même opérateur relationnel ayant un coût différent.



Question

Pourquoi ne pas systématiquement mettre en œuvre, pour un opérateur relationnel donné, la méthode de résolution ayant le coût le plus faible ?

Réponse

Parce qu'il est impossible de déterminer à l'avance, pour un opérateur relationnel donné, UNE méthode de résolution *a priori* meilleure que les autres : cela dépend du contexte de l'opérateur. De plus, chaque méthode de résolution a des prérequis pour pouvoir être mise en œuvre et ceux-ci ne sont pas forcément toujours atteints.

Le coût d'une méthode de résolution et la possibilité de la mettre en œuvre dépendent notamment :

- De l'opérateur relationnel considéré bien sûr (on sent bien intuitivement que le calcul d'une sélection n'a rien à voir avec celui d'une projection),
- De la place disponible en mémoire cache pour l'exécution de cette méthode de résolution,
- D'éventuelles contraintes pesant sur l'expression de l'opérateur (par exemple sur la forme de la condition d'une sélection ou d'une jointure),
- De la présence d'index sur la ou les relations sur lesquelles portent l'opérateur relationnel et du regroupement de ces index avec la relation sur laquelle ils portent,
- ...



Exemple

Ainsi :

- **Pour la sélection :**
 - Boucle (parcours séquentiel),
 - Utilisation d'un index primaire pour retrouver un n-uplet dont la clé est donnée ou appartient à un intervalle donné,
 - Utilisation d'un index secondaire pour retrouver un ensemble de n-uplets dont la clé est donnée ou appartient à un intervalle donné,
 - ...
- **Pour la projection :**
 - Boucle,
 - Tri puis boucle si les doublons doivent être éliminés,
 - ...
- **Pour la jointure :**
 - Si la jointure est faite sur un critère quelconque : boucles imbriquées, ...
 - S'il s'agit d'une équi-jointure :
 - S'il n'y a pas d'index sur les constituants de la jointure :
 - Boucles imbriquées,
 - Tri-fusion,
 - Hachage : on construit un index à accès par hachage sur une des deux relations et on se retrouve dans le cas suivant,
 - ...
 - S'il y a un index sur l'un des constituants de la jointure : boucle indexée,
 - ...
 - S'il y a des index triés sur les deux constituants de la jointure : fusion des feuilles des deux index.
 - ...

9.1.2.1 Notion de plan d'exécution

Il existe donc différentes méthodes de résolution de chaque opérateur relationnel apparaissant dans un arbre de requête. Le choix d'une méthode de résolution spécifique pour chacun des opérateurs relationnels apparaissant dans un arbre de requête est un **plan d'exécution**.



Définition : « plan d'exécution »

Un **plan d'exécution** est l'association d'un arbre de requête et d'une méthode de résolution donnée pour chacun des opérateurs relationnels apparaissant en son sein.

Remarque

On parle bien d'identifier une méthode de résolution particulière pour chaque opérateur relationnel de l'arbre et NON pour chaque type d'opérateur relationnel présent dans l'arbre !



Admettons que l'on sache calculer chacun des 3 types d'opérateurs relationnels de plusieurs façons : 3 méthodes de calcul d'une sélection (MSelA, MSelB et MSelC), 2 méthodes de calcul d'une projection (MProjA et MProjB) et 3 méthodes de calcul d'une jointure (MJoinA, MJoinB et MJoinC). En imaginant que toutes ces méthodes de résolution soient applicables tout le temps (*i.e.* que leurs préconditions soient tout le temps satisfaites), un arbre de requête qui contient 2 sélections, 2 jointures et 1 projection peut, en théorie, être associé à 162 plans d'exécution (3 façons de calculer la 1^{ère} sélection × 3 façons de calculer la 2^{nde} sélection × 3 façons de calculer la 1^{ère} jointure × 3 façons de calculer la 2^{nde} jointure × 2 façons de calculer la projection = 162 plans d'exécution).

9.1.2.2 Coût d'un plan d'exécution

Chaque méthode de résolution d'un opérateur relationnel ayant un coût propre, implicitement chaque plan d'exécution a un coût qui lui est propre. **Ainsi, un même arbre de requête peut être associé à plusieurs coûts de mise en œuvre : un pour chaque plan d'exécution que l'on peut lui associer.**



Exemple (début)

Soit l'arbre de requête suivant (en admettant qu'on sache ici résoudre une sélection de 2 façons, une jointure de 3 façons et une projection de 2 façons) :

- **En théorie**, on aurait donc pu associer 24 plans d'exécution à cet arbre de requêtes (2 méthodes de résolution de la sélection S1 × 3 méthodes de résolution de la jointure J1 × 2 méthodes de résolution de la sélection S2 × 2 méthodes de résolution de la projection P1).
- **En pratique**, il s'avère que ces méthodes de résolution ne sont pas toutes utilisables ici (les préconditions de certaines ne sont pas remplies dans cet arbre) et cet arbre ne peut, en réalité, être associé « qu'à » 4 plans d'exécution (2 méthodes de résolutions utilisables pour la sélection S1 × 2 méthodes de résolution utilisables pour la jointure J1 × 1 méthode de résolution utilisable pour la sélection S2 × 1 méthode de résolution utilisable pour la projection P1).



Exemple (suite et fin)

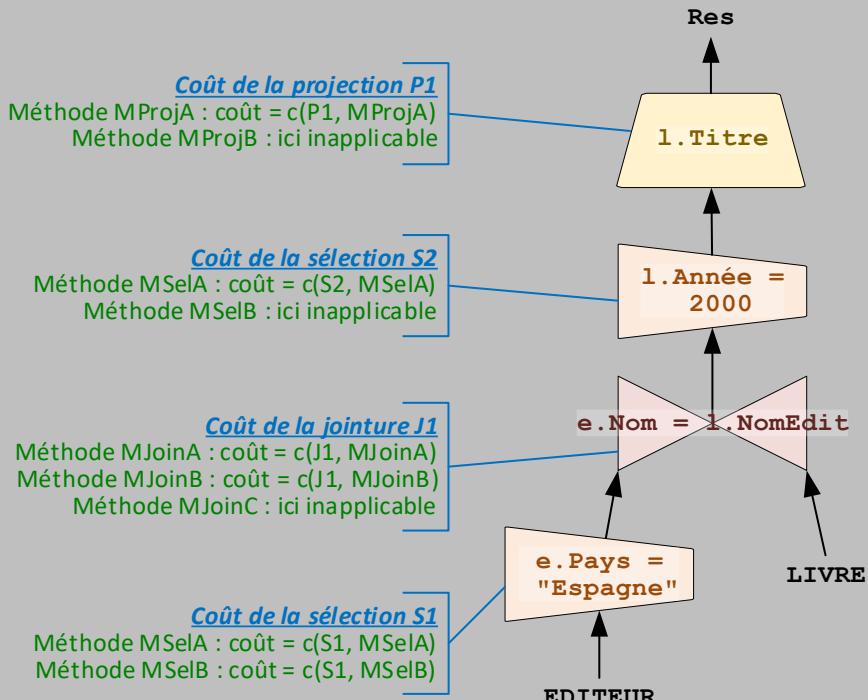


Figure 148. Méthodes de résolution et plans d'exécution d'un arbre de requête

Le coût d'un plan d'exécution est simplement la somme du coût de la mise en œuvre de chacun de ses opérateurs relationnels via la méthode de résolution qui lui a été associée dans ce plan d'exécution.

$$Coût_{planExec}(PE) = \sum_{i=1}^{i \leq Nb_{opRel}(PE)} Coût_{opRel}(opRel_i, methRes(PE, opRel_i))$$

Équation 45. Coût d'un plan d'exécution



Exemple

En reprenant l'exemple précédent (cf. Figure 148), les 4 plans d'exécution que l'on peut en pratique associer à l'arbre de requête considéré sont les suivants :

Plan d'exécution	Méthode de résolution utilisée pour le calcul...			
	...de la sélection S1	...de la jointure J1	...de la sélection S2	...de la projection P1
PE1	MSelA	MJoinA	MSelA	MProjA
PE2	MSelB	MJoinA	MSelA	MProjA
PE3	MSelA	MJoinB	MSelA	MProjA
PE4	MSelB	MJoinB	MSelA	MProjA

Tableau 52. Plans d'exécution et coûts

Le coût du plan d'exécution PE1 est donc égal à $c(S1, MSelA) + c(J1, MJoinA) + c(S2, MSelA) + c(P1, MProjA)$. On peut calculer de même le coût des plans d'exécution PE2 à PE4 (et on ne « retiendra » bien sûr que celui dont le coût est minimal).



En pratique

Le coût d'un plan d'exécution est estimé en nombre de transferts de pages ou en temps.



Rappel

La recherche des plans d'exécution (et le calcul du coût associé) est à faire pour chaque arbre de requête équivalent. Finalement, la requête sera évaluée par le plan d'exécution de moindre coût parmi tous les plans d'exécution associés à tous les arbres de requête équivalents qu'on a pu construire par rapport à elle.

9.1.2.2.1 Coût d'une méthode de résolution d'un opérateur relationnel

On va chercher à estimer le coût d'un opérateur relationnel $R = op(R_1, \dots, R_n)$ où op est un opérateur relationnel (sélection, projection ou jointure), (R_1, \dots, R_n) sont les relations d'entrée et R est la relation produite. **Nous partons du principe que chaque opérateur relationnel lit depuis la mémoire cache ses relations en entrée et écrit dans la mémoire cache son résultat (i.e. la relation produite).**

Pour simplifier, on suppose :

- Que la mémoire cache est vidée avant/après chaque évaluation d'un opérateur relationnel (**sauf les éventuelles pages qui y sont « punaisées »**),
- Dans un premier temps que le SGBD évalue un arbre de requête sans pipeline (cf. §9.2), i.e. que la relation produite par un opérateur relationnel de l'arbre est écrite en mémoire de stockage (puisque la mémoire cache est vidée avant/après chaque évaluation d'un opérateur relationnel) puis est lue depuis la mémoire de stockage via la mémoire cache par l'opérateur relationnel (s'il y en a un) situé immédiatement après dans l'arbre de requête,
- Toujours dans un premier temps que les index ne sont pas propagés au-delà des opérateurs relationnels (i.e. que les relations produites par les opérateurs relationnels ne sont pas indexées et ce même si les relations utilisées en entrée l'étaient, cf. §9.1.2.2.2.3).

Ces hypothèses étant admises, de façon générale, le coût de l'évaluation d'un opérateur relationnel peut se décomposer en deux parties distinctes :

- Le coût de la lecture des relations d'entrée depuis la mémoire de stockage (via la mémoire cache, celle-ci ayant été préalablement vidée) : c'est ce qu'on appelle le **coût de production** de la relation résultat R ,
- Le coût d'écriture du résultat en mémoire de stockage (via la mémoire cache mais celle-ci est vidée) : c'est ce qu'on appelle le **coût d'écriture** de la relation résultat R .



Définition : « coût de production (d'un opérateur relationnel) »

Le **coût de production** d'un opérateur relationnel correspond au nombre/temps de transferts de pages (en lecture, donc) qu'il doit faire pour lire son (ses) entrée(s) depuis la mémoire de stockage (via la mémoire cache, celle-ci ayant été préalablement vidée selon nos hypothèses). Le **coût de production dépend de la méthode de résolution choisie pour calculer l'opérateur relationnel**.



Définition : « coût d'écriture (d'un opérateur relationnel) »

Le **coût d'écriture** d'un opérateur relationnel correspond au nombre/temps de transferts de pages (en écriture, donc) qu'il doit faire pour écrire son résultat en mémoire de stockage (*via* la mémoire cache mais celle-ci est vidée selon nos hypothèses). **Le coût d'écriture est indépendant de la méthode de résolution choisie pour calculer l'opérateur relationnel.**

Fort logiquement, le coût de la mise en œuvre d'un opérateur relationnel *via* une méthode de résolution qui lui a été associée dans un plan d'exécution résulte de la somme de son coût de production et de son coût d'écriture :

$$Coût_{opRel}(op, methRes(op, PE)) = Coût_{production}(op, methRes(op, PE)) + Coût_{écriture}(op)$$

Équation 46. Coût d'un opérateur relationnel (mis en œuvre par une méthode donnée)

9.1.2.2.2 Paramètres d'évaluation des coûts

Les coûts de production et d'écriture dépendent, selon les cas, d'un ensemble de paramètres dont certains sont généraux (de la machine, de l'OS, du SGBD), d'autres sont liés aux relations manipulées ainsi qu'aux éventuels index associés et d'autres encore sont spécifiques à la requête.



Attention

Il n'est PAS utile de connaître tous ces paramètres pour pouvoir évaluer un coût :

- Certains de ces paramètres peuvent être calculés (donc retrouvés) à partir d'autres paramètres,
- Certains sont inutiles selon le coût que l'on a à évaluer.

9.1.2.2.2.1 Paramètres généraux (machine, OS, SGBD)

Ces paramètres donnent des indications relatives aux caractéristiques de la mémoire de stockage, à la taille d'une page et à celle de la mémoire cache notamment...

Notation	Unité(s) usuelle(s)	Paramètre	Dépendance
$tpsl_{lecture}^{fixe}$	ms	Temps de lecture d'une page (considéré fixe).	Machine
$tpse_{écriture}^{fixe}$	ms	Temps d'écriture d'une page (considéré fixe).	Machine
T_{page}^{fixe}	octets	Taille d'une page (forcément fixe).	OS/SGBD
T_{idPage}^{fixe}	octets	Taille d'un identificateur de page (forcément fixe).	OS/SGBD
$T_{caractère}^{fixe}$	octets	Taille d'un caractère (forcément fixe).	OS/SGBD
$T_{caseRep}^{fixe}$	octets	Taille (forcément fixe) d'une case du répertoire des déplacements.	SGBD
$T_{indCase}^{fixe}$	octets	Taille (forcément fixe) de l'indice d'une case du répertoire des déplacements.	SGBD
$T_{drapeau}^{fixe}$	octets	Taille (forcément fixe) du drapeau indiquant si on est sur un n-uplet ou sur un pointeur de suivi (si en mode d'adressage direct)	SGBD
$T_{idLogNU}^{fixe}$	octets	Taille (forcément fixe) d'un identificateur logique de n-uplet (si en mode d'adressage indirect).	SGBD
$T_{TvalAtt}^{fixe}$	octets	Taille (forcément fixe) de la taille d'une valeur d'un attribut (si en stockage variable des valeurs d'attributs).	SGBD
$T_{memCache}^{fixe}$	Nombre de cases	Taille de la mémoire cache.	SGBD
-	-	Stratégie d'adressage des n-uplets (directe ou indirecte).	SGBD
-	-	Stratégie de stockage des valeurs d'attributs (fixe ou variable).	SGBD
-	-	Stratégie de gestion du répertoire des déplacements (statique, avec indication du nombre fixes de cases dans ce cas, ou dynamique).	SGBD
-	-	Ordonnancement des relations pour l'arbre initial.	SGBD
-	-	Heuristiques d'optimisation mises en œuvre.	SGBD
-	-	Support de pipelines d'évaluation (et nombre, cf. §9.2).	SGBD
-	-	Possibilité de propagation des index (aucune, totale ou partielle avec liste des opérateurs relationnels concernés, cf. §9.1.2.2.2.3)	SGBD

Tableau 53. Paramètres généraux (machine, OS, SGBD)

9.1.2.2.2.2 Paramètres d'une relation R

Chaque relation R d'entrée d'un opérateur relationnel peut être caractérisée par tout ou partie des paramètres exprimés ci-dessous :

- *Paramètres relatifs aux constituants/attributs d'une relation R :*

Notation	Unité(s) usuelle(s)	Paramètre
$Nb_{att}(R)$	Nombre d'attributs	Nombre (forcément variable ¹⁹⁵) d'attributs contenus dans la relation R .
$T_{val}^{fixe moy min max}(R, X)$	Octets ou nombre de caractères	<p>Taille (fixe ou moyenne ou minimale ou maximale selon le cas) des valeurs du constituant X de la relation R.</p> <div style="background-color: #f0f0f0; padding: 5px;"> Remarque <u>En l'absence de cette propriété, et si on dispose de la taille des n-uplets de la relation R, on considère alors que les valeurs des attributs de la relation R ont toutes la même taille :</u> $\forall att \in R,$ $T_{val}^{fixe moy min max}(R, att) = \frac{T_{NU}^{fixe moy min max}(R)}{Nb_{att}(R)}$ </div>
$Nb_{valDiff}(R, X)$	Nombre de valeurs différentes	<p>Nombre (forcément variable) de valeurs différentes du constituant X de la relation R.</p> <div style="background-color: #f0f0f0; padding: 5px;"> Remarque Si le constituant X est la clé primaire de la relation R, alors on a : $Nb_{valDiff}(R, X) = Card(R)$ </div> <div style="background-color: #f0f0f0; padding: 5px;"> Attention La présence de ce paramètre implique que l'on suppose que les différentes valeurs du constituant X sont distribuées de façon équiprobable¹⁹⁶ sur les n-uplets de la relation R. </div>

Tableau 54. Paramètres relatifs aux constituants/attributs d'une relation R

¹⁹⁵ Puisqu'on peut modifier le schéma relationnel de la relation R , i.e. lui ajouter et/ou lui supprimer des attributs... Cela dit, ces modifications d'ajout/suppression d'attributs restent relativement rares !

¹⁹⁶ Une telle répartition équiprobable est homogène sur les n-uplets de la relation. Ainsi, si j'ai x valeurs v_1 à v_x réparties équiprobablement sur une relation qui compte N n-uplets, je supposerai alors avoir N/x n-uplets associés à la valeur v_1 , N/x associés à la valeur v_2 , ..., et enfin N/x associés à la valeur v_x .

- Paramètres relatifs aux n-uplets d'une relation R :

Notation	Unité(s) usuelle(s)	Paramètre
$Card(R)$ ou $Nb_{NU}(R)$	Nombre de n-uplets	Cardinalité (forcément variable) de la relation R (nombre de n-uplets contenus dans la relation R).
$T_{donneesNU}^{fixe moy min max}(R)$	octets	<p>Taille (fixe ou moyenne ou minimale ou maximale selon le cas) des données « brutes »¹⁹⁷ des n-uplets de la relation R.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> Remarque <u>En l'absence de cette propriété, et si on dispose de la taille de chaque attribut de R, on peut retrouver la taille des données « brutes » des n-uplets :</u> $T_{donneesNU}^{fixe moy min max}(R) = \sum_{i=0}^{i \leq Nb_{att}(R)} T_{val}^{fixe moy min max}(R, att_i)$ </div>
$Nb_{NU/page}^{fixe moy min max}(R)$	Nombre de n-uplets par page	<p>Nombre (fixe ou moyen ou minimal ou maximal) de n-uplets de la relation R par page de stockage des n-uplets.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> Remarque <u>En l'absence de cette propriété, on doit pouvoir la retrouver :</u> $Nb_{NU/page}^{fixe moy min max}(R) \leqq \frac{T_{utileDsPageNU}^{fixe}(*)}{T_{totaleNU}^{fixe moy min max}(R)}$ </div>
$Taux_{NUDéplacés}^{fixe moy min max}(R)$	%	<p><u>Si en mode d'adressage direct</u>, taux de n-uplets qui ne sont plus présents dans leur page d'origine. On considère alors que les pointeurs de suivi associés sont répartis équiprobablement sur les pages de stockage de la relation.</p>

Tableau 55. Paramètres relatifs aux n-uplets d'une relation R

¹⁹⁷ La taille des métadonnées des n-uplets, et donc la taille totale des n-uplets, dépend du mode d'adressage choisi (direct ou indirect) ainsi que de la stratégie de gestion du répertoire des déplacements (statique ou dynamique) et du format retenu pour le stockage des valeurs d'attributs (fixe ou variable). Se reporter au chapitre 2 (« Organisation physique des données ») et au TD associé pour de plus amples explications...

- Paramètres relatifs à la globalité d'une relation R :

Notation	Unité(s) usuelle(s)	Paramètre
$Nb_{pages}^{fixe moy min max}(R)$	Nombre de pages	<p>Nombre de pages (fixe ou moyen ou minimal ou maximal) nécessaires au stockage des données de la relation R (i.e. au stockage de ses n-uplets et de « ce qui les accompagne » en fonction du mode d'adressage choisi : éventuels pointeurs de suivi ou table de correspondance).</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> Remarque <u>En l'absence de cette propriété</u>, on doit pouvoir la retrouver : $\frac{Card(R)}{Nb_{NU/page}^{fixe moy min max}(R)}$ </div>

Tableau 56. Paramètres relatifs à la globalité d'une relation R

9.1.2.2.2.3 Paramètres d'un index I et propagation

Au niveau des index, on a besoin d'en connaître le coût d'usage et la stratégie de groupement avec la relation R indexée.

Notation	Unité(s) usuelle(s)	Paramètre
$Coût_{usage}^{fixe moy min max}(I)$	Nombre de transferts de pages (en lecture)	Coût d'usage de l'index I (i.e. nombre, fixe ou moyen ou minimal ou maximal, de transferts de pages en lecture pour retrouver 1 entrée d'index parmi toutes celles qu'il contient ¹⁹⁸).
$Nb_{pagesEI}^{fixe moy min max}(I)$	Nombre de pages	Nombre (fixe ou moyen ou minimal ou maximal) de pages de l'index I contenant des entrées d'index.
-	-	Forme de l'index (arbre B+, IMCAHSaR, IMCAHSsR ou IMCAHD)
-	-	Stratégie de groupement de l'index I avec la relation R indexée (groupé ou non-groupé).
		Rappel On groupe au plus 1 index avec une relation. L'index groupé est forcément secondaire.

Tableau 57. Paramètres relatifs à un index I

¹⁹⁸ Ou savoir qu'elle n'existe pas si jamais elle n'existe effectivement pas.

Rappel

Lorsqu'un index \mathbb{I} défini sur le constituant c d'une relation R est groupé avec cette relation R alors, dans les pages de stockage des n-uplets de cette relation, ces n-uplets sont forcément stockés « côté à côté » s'ils possèdent la même valeur v pour le constituant indexé c :

Page p	Page p'	Page p''
NU($c = v_1$)	NU($c = v_1$)	NU($c = v_7$)
NU($c = v_1$)	NU($c = v_1$)	NU($c = v_7$)
NU($c = v_1$)	NU($c = v_2$)	NU($c = v_7$)
	NU($c = v_2$)	
NU($c = v_2$)	NU($c = v_2$)	NU($c = v_8$)
NU($c = v_2$)	NU($c = v_3$)	NU($c = v_8$)
...

Figure 149. Répartition ordonnée des n-uplets quand R est indexée par un index groupé¹⁹⁹

Dans ce cas, le coût (en nombre de transferts de page) du chargement depuis les pages de stockage de la relation R de tous les n-uplets tels que $c = v$:

$$\text{Coût}_{\text{chargement } NU(R,c=v)}(TP_{\text{lecture}}) \xrightarrow{\text{ }} \frac{Nb_{\text{pages}}^{\text{fixe|moy|min|max}}(R)}{Nb_{\text{valDiff}}(R,c)}$$

Équation 47. Coût du chargement des n-uplets $c = v$ quand l'index $\mathbb{I}(R, c)$ est groupé



Quand cet index n'est pas groupé avec la relation indexée R , on considère que les n-uplets de même valeur v pour le constituant indexé c sont répartis de façon équiprobable dans les pages de stockage de la relation R :

Page p	Page p'	Page p''
NU($c = v_2$)	NU($c = v_3$)	NU($c = v_7$)
NU($c = v_8$)	NU($c = v_6$)	NU($c = v_7$)
NU($c = v_3$)	NU($c = v_2$)	NU($c = v_8$)
NU($c = v_7$)	NU($c = v_1$)	NU($c = v_2$)
NU($c = v_5$)	NU($c = v_7$)	NU($c = v_3$)
NU($c = v_1$)		NU($c = v_7$)
...

Figure 150. Répartition équiprobable des n-uplets quand R est indexée par un index non-groupé

Dans ce cas, le coût (en nombre de transferts de page) du chargement depuis les pages de stockage de la relation R de tous les n-uplets tels que $c = v$:

$$\text{Coût}_{\text{chargement } NU(R,c=v)}(TP_{\text{lecture}}) \xrightarrow{\text{ }} \frac{\text{Card}(R)}{Nb_{\text{valDiff}}(R,c)}$$

Équation 48. Coût du chargement des n-uplets $c = v$ quand l'index $\mathbb{I}(R, c)$ est non-groupé

¹⁹⁹ Forcément, en supposant ici que $v_1 < v_2 < v_3 < v_4 < v_5 < v_6 < v_7 < v_8$.

Selon les possibilités du SGBD, un index I peut être (ou non) propagé au-delà d'opérateurs relationnels. En considérant une relation R indexée sur son constituant c par un index I , cette notion de propagation peut se résumer comme suit lorsque la relation R « entre » dans un opérateur relationnel quelconque op :

- Si l'index n'est pas propagé au-delà de l'opérateur relationnel, alors le résultat délivré par cet opérateur relationnel op n'est pas indexé, que la relation R d'entrée soit indexée ou non,
- Si l'index est propagé au-delà de l'opérateur relationnel ET que la relation R est indexée sur son constituant c ET que ce constituant c existe dans le résultat de l'opérateur relationnel op , alors un index sur cette relation résultat est automatiquement construit sur le constituant c .



Définition : « propagation d'index »

La **propagation d'index** est un mécanisme qui permet, lorsqu'il est supporté, d'indexer automatiquement la relation produite à l'issue d'un opérateur relationnel de la même façon que ses relations d'entrée (lorsque cela est possible, bien sûr).



En pratique

Tous les SGBD ne supportent pas la propagation d'index. Plus encore, ceux qui supportent cette notion de propagation ne savent pas forcément la gérer au regard de tous les opérateurs relationnels. Ainsi, certains SGBD savent propager les index uniquement au-delà des sélections, d'autres uniquement au-delà des projections, ...

Le ou les opérateurs relationnels au-delà desquels un SGBD supporte la propagation sont indiqués dans sa documentation...



Question

Et à quoi ça sert ?

Réponse

On l'a déjà vu, la définition d'un index permet d'accélérer la recherche de n-uplets dans la relation indexée. Certaines méthodes de résolution d'opérateurs relationnels savent tirer parti de l'existence d'un index pour offrir un coût amélioré. La propagation des index permet donc d'optimiser le coût des opérateurs. Mais propager les index introduit une nouvelle complexité, c'est pourquoi elle n'est pas supportée tout le temps ou alors pas forcément pour tous les opérateurs relationnels.

Les avantages et inconvénients de ce mécanisme de propagation au-delà des opérateurs relationnels sont résumés ci-dessous :

Propagation des index	Avantages	Inconvénients
Lorsqu'elle est supportée	Améliore les coûts des opérateurs relationnels	Complexifie la gestion des relations produites par les opérateurs relationnels (et cette complexification a elle-même un coût)
Lorsqu'elle n'est pas supportée	Allège au maximum la gestion des relations produites par les opérateurs relationnels	Ne permet pas de bénéficier au mieux des méthodes de résolution optimales des opérateurs relationnels

Tableau 58. Avantages et inconvénients de la propagation des index

9.1.2.2.2.4 Paramètres spécifiques à une requête

Enfin, une requête contenant des conditions (aussi bien de sélection que de jointure) possède des paramètres liés à ces conditions :

Notation	Unité(s) usuelle(s)	Paramètre
$F_{sel}(R, cond)$	Taux de n-uplets	<p>Facteur de sélectivité d'une condition de sélection (<i>i.e.</i> taux de n-uplets de R vérifiant la condition de sélection $cond$).</p> <p>Remarque Le facteur de sélectivité d'une condition de sélection peut être donné quelle que soit la forme de cette condition. <u>S'il n'est pas donné ET qu'il s'agit d'une équi-sélection (<i>i.e.</i> si la condition de sélection est de la forme $att = valeur$)</u>, on supposera alors que la valeur « cherchée » existe bien et que les valeurs de l'attribut att sont réparties équiprobablement, d'où :</p> $F_{sel}(R, cond) = \frac{1}{Nb_{valDiff}(R, att)}$
$F_{join}(P, Q, cond)$	Taux de n-uplets	<p>Facteur de sélectivité d'une condition de jointure (<i>i.e.</i> taux de n-uplets du produit cartésien $P \times Q$ vérifiant la condition de jointure $cond$).</p> <p>Rappel On rappelle la cardinalité d'un produit cartésien : $Card(P \times Q) = Card(P) \times Card(Q)$</p> <p>Remarque Le facteur de sélectivité d'une condition de jointure peut être donné quelle que soit la forme de cette condition. <u>S'il n'est pas donné ET qu'il s'agit d'une équi-jointure (<i>i.e.</i> si la condition de jointure est de la forme $att_P = att_Q$)</u>, on supposera alors que cette égalité est bien vérifiée et que les valeurs des attributs att_P et att_Q sont réparties équiprobablement dans les relations P et Q, d'où :</p> $F_{join}(P, Q, cond) = \frac{1}{\max(Nb_{valDiff}(P, att_P), Nb_{valDiff}(Q, att_Q))}$

Tableau 59. Paramètres spécifiques à une requête

9.1.2.2.3 Autres notations utilisées

En plus de tout ou partie des paramètres indiqués ci-dessus, on utilisera les notations suivantes (notamment dans les pseudo-algorithmes présentant les méthodes de résolution des opérateurs relationnels) :

- $\text{cond}(t)$ (utilisé pour une sélection sur une relation R ou pour une jointure dont la condition est calculée sur la relation temporaire $R = P \times Q$) : t étant un n-uplet de la relation R , si t vérifie la condition de sélection ou de jointure cond , alors $\text{cond}(t)$ est vrai sinon $\text{cond}(t)$ est faux,
- $t \text{ conc } t'$ (utilisé pour une jointure dans laquelle une relation P et une relation Q apparaissent) : t étant un n-uplet de la relation P et t' un n-uplet de la relation Q alors $t \text{ conc } t'$ est un n-uplet résultant de la concaténation de t et de t' ,
- $t.A$ (utilisé pour une sélection, une projection ou une jointure dans laquelle une relation R apparaît) : t étant un n-uplet de la relation R et A un constituant de la relation R , $t.A$ désigne la valeur du constituant A pour le n-uplet t .

9.1.3 Coût de production des opérateurs relationnels

Nous l'avons déjà dit, le coût de production des opérateurs relationnel dépend essentiellement du type de l'opérateur considéré (bien sûr) mais aussi de la méthode de résolution choisie pour le calculer. Pour chaque type d'opérateur relationnel (sélection ou projection ou jointure), il existe « pas mal » de façons de réaliser le calcul de son résultat. Nous allons en présenter ici quelques-unes et indiquer leur coût...



Rappel

Le coût de production d'un opérateur relationnel n'a trait qu'à la lecture des pages des relations d'entrée (il ne « s'occupe » donc pas de l'écriture des pages de la relation résultat) !

9.1.3.1 Coût de production d'une sélection

La sélection est un opérateur relationnel prenant une relation R en entrée et fournissant en sortie une relation temporaire S . Une sélection est toujours réalisée par rapport à une condition cond portant sur un constituant de la relation d'entrée R et l'idée est la suivante : on copie dans la relation de sortie S tous les n-uplets de la relation d'entrée R qui vérifient la condition cond .

Notation	Portion d'arbre de requête associée
$S = \text{sel}(R, \text{cond})$	<pre> graph TD R[R] --> cond[cond] S[S] --> cond </pre>

Tableau 60. Notation et portion d'arbre de requête associée à une sélection

Nous allons étudier 2 méthodes de résolution d'une sélection...

9.1.3.1.1 Évaluation d'une sélection au moyen d'un parcours séquentiel

Cette méthode consiste simplement à parcourir tous les n-uplets de la relation d'entrée R et à « ne retenir » en sortie dans la relation résultat S que ceux qui vérifient la condition cond de la sélection.



Données : prérequis de cette méthode de résolution

Pour pouvoir être mise en œuvre, cette méthode de résolution ne nécessite que d'avoir 2 cases disponibles dans la mémoire cache :

- Une de ces cases recevra une à une les pages de la relation d'entrée R,
- La seconde case recevra une à une les pages de la relation résultat S.



En pratique

Pseudo-algorithme SélectionParcoursSéquentiel

Données d'entrée

R : la relation d'entrée

cond : la condition de la sélection

Données de sortie

S : la relation résultat

Données intermédiaires

p : une page de la relation d'entrée R

t : un n-uplet contenu dans p

Début

Pour chaque page p de la relation R **faire**

 Transférer p dans la case du cache dédiée à R

Pour chaque n-uplet t contenu dans p **faire**

Si cond(t) **alors**

 Écrire t dans la case du cache dédiée à S

FinSi

FinPour

FinPour

Fin

Le coût de production lié à cette méthode de résolution est simple à trouver, chaque page de la relation d'entrée R étant lue 1 et 1 seule fois :

$$Coût_{production}(sel(R, cond), parcoursSeq) = Nb_{pages}(R)$$

Équation 49. Coût de production d'une sélection évaluée par un parcours séquentiel

9.1.3.1.2 Évaluation d'une équi-sélection au moyen d'un parcours indexé

Cette méthode de résolution ne fonctionne que pour les sélections dont la condition de sélection est de la forme $att = valeur$. Elle utilise l'index défini sur l'attribut att pour trouver dans la relation d'entrée R rapidement les n-uplets vérifiant la condition de sélection et les « retenir » en sortie dans la relation résultat S .



Données : prérequis de cette méthode de résolution

Pour pouvoir être mise en œuvre, cette méthode de résolution nécessite :

- D'avoir 3 cases disponibles dans la mémoire cache :
 - La première case recevra une à une les pages de la relation d'entrée R ,
 - La deuxième case recevra une à une les pages de la relation résultat S ,
 - La troisième case servira au chargement des pages de l'index I .
- Que la condition de sélection soit une égalité $A = v$, où A est un attribut de la relation d'entrée R et v une valeur,
- Que la relation d'entrée R soit indexée (index I) sur son attribut A .

En pratique

Pseudo-algorithme SélectionParcoursIndexé

Données d'entrée

R : la relation d'entrée

I : l'index défini sur l'attribut A de R

v : la valeur indiquée dans la condition de sélection

Données de sortie

S : la relation résultat

Données intermédiaires

$adrLog$: une adresse logique d'un n-uplet de R

p : une page de la relation d'entrée R

Début

Chercher dans I l'entrée d'index

$(v, \{adrLog_1, \dots, adrLog_k\})$ associée à la valeur v^{200}

Pour $adrLog \in \{adrLog_1, \dots, adrLog_k\}$ **faire**

Transférer dans la case dédiée à R du cache la page p de R , si elle n'y est pas déjà, contenant le n-uplet d'adresse logique $adLog$

Écrire ce n-uplet d'adresse logique $adLog$ dans la case du cache dédiée à S

FinPour

Fin

On voit bien que cette méthode se déroule en 2 étapes, chacune « participant » au coût de production :

1. *Utilisation de l'index pour trouver les adresses logiques des n-uplets qui nous intéressent* : sa « participation » au coût de production est égal au coût d'usage de l'index,
2. *Changement des pages contenant ces n-uplets* : sa « participation » au coût de production est égal au nombre de pages de stockage à charger dans une relation pour récupérer tous les n-uplets possédant une même valeur du constituant indexé.

²⁰⁰ Pour effectuer cette recherche, se reporter aux algorithmes de recherche présentés pour chaque forme d'index dans le chapitre 4 (« Indexation des données »).

On a donc, pour cette méthode de résolution, un coût de production dépendant du groupement de l'index I avec la relation R d'entrée :

- Si l'index I est non-groupé :

$$\text{Coût}_{\text{production}}(\text{sel}(R, \text{cond}), \text{parcoursInd}) = \overbrace{\text{Coût}_{\text{usage}}(I)}^{\text{Recherche dans } I \text{ de l'EI } (v,a)} + \overbrace{\left(\frac{\text{Card}(R)}{\text{Nb}_{\text{valDiff}}(R, A)} \right)}^{\substack{\text{Chargement des n-uplets} \\ \text{de } R \text{ dont l'adresse logique} \\ \text{est dans l'EI } (v,a) \\ \text{quand } I \text{ est non-groupé}}} \quad ^{201}$$

Équation 50. Coût de production d'une sélection évaluée par un parcours indexé (I non-groupé)

- Si l'index I est groupé :

$$\text{Coût}_{\text{production}}(\text{sel}(R, \text{cond}), \text{parcoursInd}) = \overbrace{\text{Coût}_{\text{usage}}(I)}^{\text{Recherche dans } I \text{ de l'EI } (v,a)} + \overbrace{\left(\frac{\text{Nb}_{\text{pages}}(R)}{\text{Nb}_{\text{valDiff}}(R, A)} \right)}^{\substack{\text{Chargement des n-uplets} \\ \text{de } R \text{ dont l'adresse logique} \\ \text{est dans l'EI } (v,a) \\ \text{quand } I \text{ est groupé}}} \quad ^{202}$$

Équation 51. Coût de production d'une sélection évaluée par un parcours indexé (I groupé)

Dans les 2 cas (i.e. que l'index soit groupé ou non), on observe que plus le nombre de valeurs différentes de l'attribut apparaissant dans la condition de la sélection est grand, plus le coût du chargement des n-uplets recherchés dans la relation d'entrée R est faible... Cette méthode de résolution fonctionne donc d'autant mieux (au niveau de ses performances) que la condition de sélection porte sur un attribut ayant un grand nombre de valeurs différentes.

9.1.3.2 Coût de production d'une jointure

La jointure est un opérateur relationnel prenant deux relations P et Q en entrée et fournissant en sortie une relation temporaire J . Une jointure est toujours réalisée par rapport à une condition `cond` portant sur un constituant de chaque relation d'entrée P et Q et l'idée est la suivante : on copie dans la relation de sortie J tous les n-uplets issus du produit cartésien des relations d'entrée $P \times Q$ qui vérifient la condition `cond`.



Remarque

Bien que la jointure soit un opérateur relationnel commutatif, les 2 relations d'entrée ne jouent pas le même rôle : cette « asymétrie » se voit très bien dans les différentes méthodes de résolution d'une jointure (voir ci-après). Dans l'arbre de requête, on représente la **relation externe** comme entrée gauche et la **relation interne** comme entrée droite (cf. Tableau 61). La lecture des 2 relations d'entrée participe au coût de production de la jointure.

Nous allons étudier 3 méthodes de résolution d'une jointure...

²⁰¹ Le résultat de la fraction est arrondi par excès AVANT d'être additionné et vaut au moins 1 si au moins 1 n-uplet de la relation d'entrée R vérifie la condition de sélection `cond` (cette fraction vaut 0 sinon).

²⁰² Là encore, le résultat de la fraction est arrondi par excès AVANT d'être additionné et vaut au moins 1 si au moins 1 n-uplet de la relation d'entrée R vérifie la condition de sélection `cond` (cette fraction vaut 0 sinon).

Notation	Portion d'arbre de requête associée
$J = \text{join}(P, Q, \text{cond})$	

Tableau 61. Notation et portion d'arbre de requête associée à une jointure

9.1.3.2.1 Évaluation d'une jointure au moyen de boucles imbriquées

Avec cette méthode d'évaluation, la plus « basique » de toutes, on parcourt, en la chargeant page après page, tous les n-uplets de la relation externe P et, pour chacun d'entre eux, on le concatène, en la chargeant page par page, à tous les n-uplets de la relation interne Q . Si cette concaténation respecte la condition de la jointure cond , elle est écrite dans la relation résultat J .

En pratique

Pseudo-algorithme JointureBouclesImbriquées

Données d'entrée

- P : la relation externe
- Q : la relation interne
- cond : la condition de la jointure

Données de sortie

- J : la relation résultat

Données intermédiaires

- p : une page de la relation externe P
- q : une page de la relation interne Q
- t_p : un n-uplet de P contenu dans p
- t_q : un n-uplet de Q contenu dans q

Début

- Pour** chaque page p de P **faire**
- Transférer p dans la case dédiée du cache
- Pour** chaque page q de Q **faire**
- Transférer q dans la case dédiée du cache
- Pour** chaque n-uplet t_q de q **faire**
- Pour** chaque n-uplet t_p de p **faire**
- Si** $\text{cond}(t_p \text{ conc } t_q)$ **alors**
- Écrire le n-uplet $t_p \text{ conc } t_q$ dans la case du cache dédiée à J
- FinSi**
- FinPour**
- FinPour**
- FinPour**
- Fin**





Données : prérequis de cette méthode de résolution

Pour pouvoir être mise en œuvre, cette méthode de résolution ne nécessite que d'avoir 3 cases disponibles dans la mémoire cache :

- Une de ces cases recevra une à une les pages de la relation externe P ,
- La deuxième de ces cases recevra une à une les pages de la relation interne Q ,
- La dernière case recevra une à une les pages de la relation résultat J .

On voit ici qu'on lit 1 et 1 seule fois chaque page de la relation externe P . On voit aussi que, pour chaque page chargée de la relation externe P , on charge 1 fois chaque page de la relation interne Q .

On a donc :

$$Coût_{production}(join(P, Q, cond), bouclesImbr) = \overbrace{Nb_{pages}(P)}^{lecture de la relation externe P} + \overbrace{(Nb_{pages}(P) \times Nb_{pages}(Q))}^{lecture de la relation interne Q}$$

Équation 52. Coût de production d'une jointure évaluée par des boucles imbriquées

Cette formule du coût de production lié à cette méthode de résolution nous indique que le coût de production est minimal si on parcourt d'abord la relation « la plus petite » (i.e. si la relation externe est la relation occupant le moins de pages).

9.1.3.2.2 Évaluation d'une jointure au moyen de boucles optimisées

Cette méthode de résolution est une évolution de la précédente reposant sur l'usage du cache : au lieu de charger 1 à 1 les pages de la relation externe P (afin de « lier » leurs n-uplets à ceux de la relation interne Q qu'on peut leur « associer »), nous allons les charger par groupes de M pages (i.e. contenant un nombre M de pages, la valeur de M dépendant de la place disponible en mémoire cache que l'on peut dédier au chargement des pages de la relation externe P).



Données : prérequis de cette méthode de résolution

Pour pouvoir être mise en œuvre, cette méthode de résolution ne nécessite que d'avoir N cases disponibles dans la mémoire cache :

- Les M ($M = N - 2$) premières cases recevront les pages de la relation externe P , par groupes de M pages donc,
- Une autre case recevra une à une les pages de la relation interne Q ,
- La dernière case recevra une à une les pages de la relation résultat J .



Remarques

Plus il y aura de cases disponibles dans la mémoire cache, plus on pourra faire des grands groupes de pages de la relation externe P , plus la méthode sera optimale (l'idéal étant de pouvoir charger d'un coup dans le cache les pages de la relation externe P).

Si la mémoire cache est vide lors de la production du résultat de la jointure²⁰³, alors :

$$M = T_{memCache}^{fixe} - 2$$

²⁰³ Rappelons que, jusque-là, nous avons supposé que la mémoire cache était vidée avant l'évaluation de chaque opérateur relationnel (hormis les pages punaisées, bien sûr).



En pratique

Pseudo-algorithme JointureBouclesOptimisées

Données d'entrée

P : la relation externe

Q : la relation interne

$cond$: la condition de la jointure

N : le nombre de cases libres dans le cache

Données de sortie

J : la relation résultat

Données intermédiaires

M : le nombre de pages du cache dédiées à P

Gr_P : un groupe de pages de la relation externe P

q : une page de la relation interne Q

t_p : un n-uplet de P contenu dans Gr_P

t_Q : un n-uplet de Q contenu dans q

Début

$M \leftarrow N - 2$

Pour chaque groupe Gr_P de M pages de P **faire**

Transférer Gr_P dans les M cases dédiées du cache

Pour chaque page q de Q **faire**

Transférer q dans la case dédiée du cache

Pour chaque n-uplet t_Q de q **faire**

Pour chaque n-uplet t_p de Gr_P **faire**

Si $cond(t_p, conc\ t_Q)$ **alors**

Écrire t_p conc t_Q dans la case dédiée à J

FinSi

FinPour

FinPour

FinPour

FinPour

Fin

Concernant le chargement des pages de la relation externe P , rien ne change : on les charge toujours 1 et 1 seule fois chacune. En revanche, au lieu de charger chaque page de la relation interne Q 1 fois pour chaque page chargée de la relation externe P , on ne charge chaque page de la relation interne Q qu'1 fois pour chaque groupe de M pages de la relation externe P . On a donc :

$$Coût_{production}(join(P, Q, cond), bouclesOpt) =$$

$$\overbrace{Nb_{pages}(P)}^{lecture\ de\ la\ relation\ externe\ P} + \overbrace{\left(\frac{Nb_{pages}(P)}{M} \right)^{204} \times Nb_{Pages}(Q)}^{lecture\ de\ la\ relation\ interne\ Q}$$

Nombre de groupes de M pages dans P

Équation 53. Coût de production d'une jointure évaluée par des boucles optimisées

²⁰⁴ La fraction calculant le nombre de groupes de M pages dans la relation externe P doit être arrondie par excès AVANT d'être multipliée !!!

Là encore, cette formule du coût de production lié à cette méthode de résolution nous indique que le coût de production est minimal si on parcourt d'abord la relation « la plus petite » (*i.e.* si la relation externe est la relation occupant le moins de pages).

9.1.3.2.3 Évaluation d'une équi-jointure au moyen d'une boucle indexée

Cette méthode de résolution ne fonctionne que pour les jointures dont la condition de jointure est de la forme $att_P = att_Q$. Elle utilise l'index défini sur l'attribut att_Q pour trouver dans la relation interne Q rapidement les n-uplets vérifiant la condition de jointure pour la valeur courante de att_P .

En pratique

Pseudo-algorithme JointureBoucleIndexée

Données d'entrée

P : la relation externe

Q : la relation interne

I : l'index défini sur l'attribut att_Q de Q

Données de sortie

J : la relation résultat

Données intermédiaires

p : une page de la relation externe P

q : une page de la relation interne Q

t_p : un n-uplet de P contenu dans p

t_q : un n-uplet de Q contenu dans q

v : la valeur de l'attribut att_P de t_p

$adrLog$: une adresse logique d'un n-uplet de Q

Début

Pour chaque page p de P **faire**

Transférer dans le cache la page p

Pour chaque n-uplet t_p de p **faire**

$v \leftarrow t.att_P$

Chercher dans l'index I l'entrée d'index

$(v, \{adrLog_1, \dots, adrLog_k\})$ associée à v^{205}

Pour $adrLog \in \{adrLog_1, \dots, adrLog_k\}$ **faire**

Transférer dans le tampon, si elle n'y est pas déjà, la page q de Q contenant le n-uplet t_q d'adresse logique $adrLog$

Écrire le n-uplet t_p conc t_q dans la case du cache dédiée à J

FinPour

FinPour

FinPour

Fin



²⁰⁵ Pour effectuer cette recherche, se reporter aux algorithmes de recherche présentés pour chaque forme d'index dans le chapitre 4 (« Indexation des données »).



Données : prérequis de cette méthode de résolution

Pour pouvoir être mise en œuvre, cette méthode de résolution nécessite :

- D'avoir 4 cases disponibles dans la mémoire cache :
 - La première case recevra une à une les pages de la relation externe P ,
 - La deuxième case recevra une à une les pages de la relation interne Q ,
 - La troisième case recevra une à une les pages de la relation résultat Σ ,
 - La dernière case servira au chargement des pages de l'index I .
- Que la condition de jointure soit une égalité $att_P = att_Q$, où att_P est un attribut de la relation externe P et att_Q un attribut de la relation interne Q ,
- Que la relation interne Q soit indexée (index I) sur son attribut att_Q .

Concernant la lecture de la relation externe P , on charge, ici encore, 1 et 1 seule fois chacune de ses pages. De plus, pour chaque n-uplet qu'elles contiennent, on effectue les 2 traitements suivants :

1. *Utilisation de l'index pour trouver les adresses logiques des n-uplets de Q qui nous intéressent :* sa « participation » au coût de production est égal au coût d'usage de l'index,
2. *Chargement des pages de Q contenant ces n-uplets :* sa « participation » au coût de production est égal au nombre de pages de stockage à charger dans une relation pour récupérer tous les n-uplets possédant une même valeur du constituant indexé.

On a donc, pour cette méthode de résolution, un coût de production dépendant du groupement de l'index I avec la relation interne Q :

- Si l'index I est non-groupé :

$$Coût_{production}(join(P, Q, cond), boucleInd) =$$

$$\underbrace{Nb_{pages}(P)}_{\text{lecture de la relation externe } P} + Card(P) \times \left(\underbrace{\frac{Coût_{usage}(I)}{\text{Recherche dans } I \text{ de l'EI } (v,a)}}_{\text{lecture de la relation interne } Q} + \underbrace{\frac{Card(Q)}{Nb_{valDiff}(Q, att_Q)}}_{\substack{\text{Chargement des n-uplets de } Q \\ \text{identifiés dans EI (I non-groupé)}}} \right)^{206}$$

Équation 54. Coût de production d'une jointure évaluée par une boucle indexée (I non-groupé)

- Si l'index I est groupé :

$$Coût_{production}(join(P, Q, cond), boucleInd) =$$

$$\underbrace{Nb_{pages}(P)}_{\text{lecture de la relation externe } P} + Card(P) \times \left(\underbrace{\frac{Coût_{usage}(I)}{\text{Recherche dans } I \text{ de l'EI } (v,a)}}_{\text{lecture de la relation interne } Q} + \underbrace{\frac{Nb_{pages}(Q)}{Nb_{valDiff}(Q, att_Q)}}_{\substack{\text{Chargement des n-uplets de } Q \\ \text{identifiés dans EI (I groupé)}}} \right)^{207}$$

Équation 55. Coût de production d'une jointure évaluée par une boucle indexée (I groupé)

²⁰⁶ Le résultat de la fraction est arrondi par excès AVANT d'être additionné et vaut au moins 1 si au moins 1 n-uplet de la relation interne Q peut être joint à un n-uplet de la relation externe P (cette fraction vaut 0 sinon).

²⁰⁷ Là encore, le résultat de la fraction est arrondi par excès AVANT d'être additionné et vaut au moins 1 si au moins 1 n-uplet de la relation interne Q peut être joint à un n-uplet de la relation externe P (elle vaut 0 sinon).

Le calcul d'une jointure par boucle indexée est d'autant plus intéressant que la relation externe est de faible cardinalité par rapport à la relation interne. Dans le cas contraire, le calcul de la jointure par boucles optimisées est souvent plus performant.

9.1.3.3 Coût de production d'une projection (sans élimination des doublons)

La projection s'effectue sur une relation d'entrée R via un sous-ensemble de ses attributs $\{A_1, \dots, A_n\}$. Pour chaque n-uplet de la relation d'entrée R , la projection ne conserve en sortie que les valeurs des attributs sur lesquels la projection est réalisée.

Notation	Portion d'arbre de requête associée
$P = \text{proj}(R, \{A_1, \dots, A_n\})$	<pre> graph TD R[R] --> A[A₁, ..., A_n] A --> P[P] </pre>

Tableau 62. Notation et portion d'arbre de requête associée à une projection



Remarque

La projection peut s'effectuer avec ou sans élimination des doublons. En effet, si les attributs faisant partie de la clé primaire ne font pas tous partie des attributs sur lesquels la projection est réalisée, il est possible que des n-uplets existent en double dans la relation résultat P . La possibilité d'éliminer de tels doublons est offerte par les langages de requête, mais elle doit être explicitement mise en œuvre...

Pour simplifier, nous ne traiterons ici que les projections SANS élimination des doublons.



En pratique

En SQL, c'est l'opérateur DISTINCT²⁰⁸ qui s'occupe d'éliminer les doublons de la relation résultat d'une projection (il est donc toujours utilisé pour « compléter » l'opérateur SELECT) :

```

SELECT DISTINCT l.Auteur
FROM Livre l
WHERE l.Annee > 2000
  
```

9.1.3.3.1 Évaluation d'une projection (sans élimination des doublons) au moyen d'un parcours séquentiel

Cette méthode de résolution est la plus « basique » pour calculer une projection (sans élimination des doublons). Elle consiste à simplement parcourir la relation d'entrée R et, pour chaque n-uplet qu'elle contient, à écrire dans la relation résultat P une copie des valeurs de ce n-uplet pour les attributs sur lesquels la projection est réalisée.

²⁰⁸ On fait souvent le parallèle avec l'opérateur de groupement GROUP BY mais il n'y a pas de stricte équivalence entre eux dans tous les cas.



Données : prérequis de cette méthode de résolution

Pour pouvoir être mise en œuvre, cette méthode de résolution ne nécessite que d'avoir 2 cases disponibles dans la mémoire cache :

- Une de ces cases recevra une à une les pages de la relation d'entrée R,
- La seconde case recevra une à une les pages de la relation résultat P.



En pratique

Pseudo-algorithme ProjectionParcoursSéquentiel
Données d'entrée

R : la relation d'entrée

$SetAtt$: l'ensemble des attributs projetés

Données de sortie

P : la relation résultat

Données intermédiaires

p : une page de la relation d'entrée R

t_p : un n-uplet contenu dans p

t_{Proj} : un n-uplet du résultat P

Début

Pour chaque page p de la relation R **faire**

Transférer p dans la case du cache dédiée à R

Pour chaque n-uplet t_p de p **faire**

$t_{Proj} \leftarrow SetAtt(t_p)$

Écrire t_{Proj} dans la case du cache dédiée à P

FinPour

FinPour

Fin

Cette méthode de résolution lit simplement 1 et 1 seule fois chaque page de la relation d'entrée R . Son coût de production est donc simple à évaluer :

$$\text{Coût}_{production}(\text{proj}(R, \{A_1, \dots, A_n\}), \text{parcoursSeq}) = Nb_{pages}(R)$$

Équation 56. Coût de production d'une projection évaluée par un parcours séquentiel

9.1.3.3.2 Évaluation d'une projection (sans élimination des doublons) au moyen d'un parcours indexé

Cette méthode de résolution, *a priori* plus performante, tire partie de la présence d'un index sur le constituant formé des attributs projetés²⁰⁹. L'idée est de récupérer toutes les entrées d'index contenu dans l'index et d'écrire dans la relation résultat de la projection P des n-uplets formés à partir des valeurs de ces entrées d'index.



Remarque

Une légère modification de cette méthode de résolution permettrait d'éliminer les doublons de la projection.

²⁰⁹ Nous n'avons pas étudié les index à accès multi-critères mais cela ne nous empêche pas d'en tirer parti !



Données : prérequis de cette méthode de résolution

Pour pouvoir être mise en œuvre, cette méthode de résolution nécessite :

- D'avoir 3 cases disponibles dans la mémoire cache :
 - La première case recevra une à une les pages de la relation d'entrée R ,
 - La deuxième case recevra une à une les pages de la relation résultat P ,
 - La troisième case servira au chargement des pages de l'index I .
- Que la relation d'entrée R soit indexée (index I) sur un constituant formé de tous les attributs projetés $\{A_1, \dots, A_n\}$.



En pratique

Pseudo-algorithme ProjectionParcoursIndexé

Données d'entrée

R : la relation d'entrée

I : l'index défini sur l'attribut A de R

Données de sortie

P : la relation résultat

Données intermédiaires

EI : une entrée d'index de I

v : la valeur identifiant EI dans I

n : le nombre d'adresses logiques contenues dans EI

cpt : un entier (compteur)

Début

Pour chaque $EI \in I$ **faire**

$v \leftarrow$ valeur identifiant EI dans I

$n \leftarrow$ nombre d'adresses logiques dans EI

Pour $cpt \in \{1, \dots, n\}$ **faire**

Écrire le n -uplet v dans la case du cache de P

FinPour

FinPour

Fin



Question

Pourquoi écrire plusieurs fois le n -uplet v dans la relation résultat ?

Réponse

Précisément parce que l'on n'élimine pas les doublons ! Ainsi, s'il y avait dans la relation d'entrée R k n -uplets possédant cette valeur v pour le constituant projeté, on doit toujours en avoir k dans la relation résultat P (on en écrit donc autant que d'adresses logiques de n -uplets trouvées dans l'entrée d'index de valeur v)... Si nous avions voulu faire une projection en éliminant les doublons, il aurait suffi d'écrire dans la relation résultat P un seul n -uplet formé sur chaque valeur v du constituant indexé et projeté.

Cette méthode de résolution de charge aucune page de la relation d'entrée R ! En revanche, elle charge toutes les pages de l'index I contenant des entrées d'index. Son coût de production est donc :

$$Coût_{production}(\text{proj}(P, \{A_1, \dots, A_n\})) = Nb_{pagesEI}(I)$$

Équation 57. Coût de production d'une projection évaluée par un parcours indexé

9.1.4 Coût d'écriture du résultat des opérateurs relationnels

Nous venons d'étudier différentes méthodes de résolution des opérateurs relationnels et, pour chacune d'entre elles, nous avons identifié une formule (voire deux) permettant d'en estimer le coût de production, c'est-à-dire les pages à transférer depuis les relations d'entrée et/ou leurs index...



Question

Qu'en est-il de l'écriture du résultat de ces opérateurs ?

Réponse

Le coût d'écriture de ce résultat ne dépend pas directement, lui, ni de l'opérateur relationnel considéré ni de la méthode de résolution choisie pour le calculer !

En effet, quel que soit l'opérateur relationnel et la méthode de résolution, l'écriture du résultat se fait comme suit, via la mémoire cache :



En pratique

```

Pseudo-algorithme EcritureRésultat
Données d'entrée
    t : un n-uplet produit par l'opérateur relationnel
Données de sortie
    Res : la relation résultat de l'opérateur
Données intermédiaires
    p : une page de la relation résultat Res
    c : la case du tampon dédiée à Res
Début
    c ← page vide p
    Pour chaque n-uplet t produit par l'opérateur faire
        Si la page p est pleine alors
            Décharger la page p
            c ← nouvelle page vide p
        FinSi
        Écrire t dans la page p
    FinPour
Fin
```

Ce pseudo-algorithme, simplissime, nous fait voir que l'on réalise autant de transferts de pages en écriture au niveau de la relation résultat Res que celle-ci en contient (chacune de ses pages est écrite 1 et 1 seule fois). Le coût d'écriture d'un opérateur relationnel est donc le suivant :

$$Coût_{écriture} = Nb_{pages}(Res)$$

Équation 58. Coût d'écriture de la relation résultat d'un opérateur relationnel



Question

OK, mais combien de pages contient cette relation résultat ?

Réponse

C'est là qu'on dépend de nouveau de l'opérateur relationnel considéré... Mais PAS de la méthode de résolution choisie pour le calculer !

Évaluer la taille d'une relation résultat passe notamment par l'évaluation de la cardinalité de cette relation ET par l'évaluation de la taille des n-uplets de cette relation (à partir de celle de leurs données « brutes »). Puis, comme pour les relations « classiques », on peut estimer le nombre de n-uplets par page de stockage et, donc, le nombre de pages qu'occupe la relation résultat.



Remarque

Ne pas oublier que, pour cela, on doit prendre en compte les différentes stratégies retenues (mode d'adressage direct ou indirect, stockage fixe ou variable des valeurs d'attributs, gestion statique ou dynamique du répertoire des déplacements) et, donc, l'impact des éventuelles métadonnées associées (drapeau, taille des pointeurs de suivi, taille de la table de correspondance, taille de la taille des valeurs d'attributs, prise en compte du répertoire des déplacements, ...). Se reporter au chapitre 2 (« Organisation physique des données ») pour de plus amples informations...



Rappel

Rappelons que l'on a supposé que les différentes valeurs d'un constituant sont réparties uniformément sur les n-uplets d'une relation et que tous les attributs avaient une valeur (pas de valeur `NULL` donc).

9.1.4.1 Évaluation de la taille de la relation produite par une sélection

L'opérateur relationnel de sélection fait passer les n-uplets de la relation d'entrée dans un « filtre » (en fonction de la condition de la sélection). La cardinalité de la relation résultat est donc plus petite (au pire, égale à) que celle de la relation d'entrée. Plus précisément :

$$\text{Card}(\text{sel}(R, \text{cond})) \leq \text{Card}(R) \times F_{\text{sel}}(R, \text{cond})$$

Équation 59. Évaluation de la cardinalité de la relation résultat d'une sélection



Rappel

Le facteur de sélectivité d'une condition de sélection peut être donné quelle que soit la forme de cette condition. S'il n'est pas donné ET qu'il s'agit d'une équi-sélection (i.e. si la condition de sélection est de la forme `att = valeur`), on supposera alors que la valeur « cherchée » existe bien et que les valeurs de l'attribut `att` sont réparties équiprobablement, d'où :

$$F_{\text{sel}}(R, \text{cond}) = \frac{1}{Nb_{\text{valDiff}}(R, \text{att})}$$



Question

Et pour les sélections dont la condition est une conjonction de conditions ?

Réponse

Si on vous a donné directement le facteur de sélectivité de la condition complexe, pas de souci : il n'y a qu'à l'utiliser directement. Sinon, le facteur de sélectivité d'une condition de sélection complexe est le produit des facteurs de sélectivité des conditions simples qui la composent :

$$F_{\text{sel}}(R, \text{cond}_1 \wedge \dots \wedge \text{cond}_n) = \prod_{i=1}^{i \leq n} F_{\text{sel}}(R, \text{cond}_i)$$

Au niveau de la taille des données « brutes » des n-uplets de sa relation résultat, la sélection ne les modifie pas (elle n'ajoute ni ne supprime d'attributs). Ainsi :

$$T_{\text{donnéesNU}}^{\text{fixe|moy|min|max}}(\text{sel}(R, \text{cond})) = T_{\text{donnéesNU}}^{\text{fixe|moy|min|max}}(R)$$

Équation 60. Évaluation de la taille des données des n-uplets produits par une sélection

9.1.4.2 Évaluation de la taille de la relation produite par une jointure

L'opérateur relationnel de jointure réalise un produit cartésien de ses 2 relations d'entrée : il concatène chaque n-uplet de la relation externe avec chaque n-uplet de la relation interne. Ce produit cartésien « filtré » (dynamiquement, *i.e.* au fur et à mesure de sa création) en fonction de la condition de jointure. La cardinalité de la relation résultat issue de la jointure est donc basée sur celle du produit cartésien des 2 relations d'entrée de cette jointure.



Rappel

La cardinalité du produit cartésien $P \times Q$ est :

$$\text{Card}(P \times Q) = \text{Card}(P) \times \text{Card}(Q)$$

Partant de là, le principe de « filtrage » (par la condition de jointure) se passe de façon analogue au « filtrage » réalisé par une sélection. Ainsi :

$$\text{Card}(\text{join}(P, Q, \text{cond})) \stackrel{?}{=} \text{Card}(P) \times \text{Card}(Q) \times F_{\text{join}}(P, Q, \text{cond})$$

Équation 61. Évaluation de la cardinalité de la relation résultat d'une jointure



Rappel

Le facteur de sélectivité d'une condition de jointure peut être donné quelle que soit la forme de cette condition. S'il n'est pas donné ET qu'il s'agit d'une équi-jointure (*i.e.* si la condition de jointure est de la forme $\text{att}_P = \text{att}_Q$), on supposera alors que cette égalité est bien vérifiée et que les valeurs des attributs att_P et att_Q sont réparties équiprobablement dans les relations P et Q , d'où :

$$F_{\text{join}}(P, Q, \text{cond}) = \frac{1}{\max(Nb_{valDiff}(P, \text{att}_P), Nb_{valDiff}(Q, \text{att}_Q))}$$



Question

Et pour les jointures dont la condition est une conjonction de conditions ?

Réponse

Si on vous a donné directement le facteur de sélectivité de la condition complexe, pas de souci : il n'y a qu'à l'utiliser directement. Sinon, le facteur de sélectivité d'une condition de jointure complexe est le produit des facteurs de sélectivité des conditions simples qui la composent :

$$F_{\text{join}}(P, Q, \text{cond}_1 \wedge \dots \wedge \text{cond}_n) = \prod_{i=1}^{i \leq n} F_{\text{join}}(P, Q, \text{cond}_i)$$

Les n-uplets de la relation résultat issue d'une jointure résultent tous de la concaténation d'un n-uplet de la relation externe avec un n-uplet de la relation interne. Ainsi :

$$T_{\text{donnéesNU}}^{\text{fixe|moy|min|max}}(\text{join}(P, Q, \text{cond})) = T_{\text{donnéesNU}}^{\text{fixe|moy|min|max}}(P) + T_{\text{donnéesNU}}^{\text{fixe|moy|min|max}}(Q)$$

Équation 62. Évaluation de la taille des données des n-uplets produits par une jointure

9.1.4.3 Évaluation de la taille de la relation produite par une projection (sans élimination des doublons)

La projection copie dans la relation résultat tous les n-uplets de la relation d'entrée mais en ne conservant que les valeurs des attributs sur lesquels elle est réalisée. Ainsi :

$$\text{Card}(\text{proj}(R, \{A_1, \dots, A_n\})) = \text{Card}(R)$$

Équation 63. Évaluation de la cardinalité de la relation résultat d'une projection

En revanche, étant donné qu'on ne conserve qu'une partie des attributs de la relation d'entrée, la taille des données « brutes » des n-uplets est, elle, modifiée :

$$T_{\text{donnéesNU}}^{\text{fixe|moy|min|max}}(\text{proj}(R, \{A_1, \dots, A_n\})) = \sum_{i=1}^{i \leq n} T_{\text{val}}^{\text{fixe|moy|min|max}}(R, A_i)$$

Équation 64. Évaluation de la taille des données des n-uplets produits par une projection

9.1.5 Étude de cas (partielle)

Soit une BD décrivant des livres et leurs éditeurs. Elle contient 2 relations et son modèle conceptuel est le suivant :

LIVRE (ISBN, Titre, Auteur, NomEdit, Année, Prix)
 EDITEUR (Nom, Adresse, Pays)

On souhaite récupérer le titre de tous les livres parus en l'an 2 000 et dont la maison d'édition est en Espagne. La requête SQL correspondante est la suivante :

```
SELECT l.Titre
FROM LIVRE l, EDITEUR e
WHERE e.Nom = l.NomEdit
AND e.Pays = 'Espagne'
AND l.Année = 2000
```

Nous allons réaliser (partiellement) l'étude de l'optimisation de cette requête. Pour cela :

- Nous donnons différents paramètres (généraux, propres à chaque relation ou à la requête),
- Nous construisons l'arbre de requête initial,
- Nous élaborons les arbres de requêtes équivalents (au moins en partie),
- Nous établissons les plans d'exécution que l'on peut associer à chacun de ces arbres et évaluons son coût.

9.1.5.1 Paramètres donnés

Nous donnons ci-dessous les paramètres dont nous disposons pour réaliser l'étude de l'optimisation de la requête :

- *Paramètres généraux (machine, OS, SGBD) :*

Paramètre	Valeur donnée à ce paramètre
$tps_{lecture}^{fixe}$	1ms
$tps_{écriture}^{fixe}$	2ms
T_{page}^{fixe}	2 048 octets
T_{idPage}^{fixe}	4 octets (on a au maximum 4 294 967 296 pages)
$T_{caractère}^{fixe}$	2 octets (codage UTF-8)
$T_{caseRep}^{fixe}$	2 octets (déplacement maximal = 65 536 octets)
$T_{indCase}^{fixe}$	1 octet (256 cases au maximum dans un répertoire des déplacements)
$T_{idLogNU}^{fixe}$	4 octets (au plus 4 294 967 296 n-uplets)
$T_{TvalAtt}^{fixe}$	2 octets (valeur d'attribut sur 65 536 octets au maximum)
$T_{memCache}^{fixe}$	50 cases au total
Adressage des n-uplets	On est en mode d'adressage direct.
Format de stockage des valeurs d'attributs	On est en format variable.
Gestion du répertoire des déplacements	On a adopté une stratégie de gestion statique (le répertoire des déplacements situé à la fin des pages de stockage des n-uplets contient toujours 10 cases).
Ordonnancement des relations dans l'arbre de requête initial	Les relations sont introduites (dans la mesure de possible, i.e. en respectant les jointures apparaissant dans la requête) par ordre croissant de leur cardinalité.
Heuristiques d'optimisation supportées	On ne construit que des arbres linaires gauches de jointure (ALGJ).
Nombre de pipelines d'évaluation mis en œuvre	Aucun pipeline d'évaluation n'est mis en œuvre.
Propagation des index	Aucune propagation des index n'est effectuée.

- Paramètres des relations *LIVRE* et *EDITEUR*: tous les n-uplets sont dans leur page d'origine

Relation R	Card(R)	Attribut A	$T_{val}^{moy}(R, A)$	$Nb_{valDiff}(R, A)$
LIVRE	12 000	ISBN	15 caractères	12 000
		Titre	75 caractères	12 000
		Auteur	50 caractères	9 000
		NomEdit	50 caractères	300
		Année	4 caractères	200
		Prix	6 caractères	4 000
EDITEUR	300	Nom	50 caractères	300
		Adresse	300 caractères	250
		Pays	50 caractères	10

- Paramètres des index sur les relations *LIVRE* et *EDITEUR*: la racine de chaque arbre B+ est punaisée en mémoire cache (cela est déjà pris en compte dans le coût de leur usage ci-dessous)

Index I	Forme	Coût_{usage}(I)	Groupement de I
LIVRE (<u>ISBN</u>)	Arbre B+	2	Non
LIVRE (NomEdit)	Arbre B+	2	Oui
LIVRE (Année)	Arbre B+	1	Non
EDITEUR (<u>Nom</u>)	Arbre B+	2	Non
EDITEUR (Pays)	Arbre B+	1	Oui

9.1.5.2 Paramètres calculés

Nous allons devoir, à partir des paramètres précédents, en calculer d'autres pour réaliser l'étude de l'optimisation...

- Pour les paramètres calculés liés aux relations *LIVRE* et *EDITEUR*:

Paramètre calculé (début)	Relation R	
	LIVRE	EDITEUR
$T_{donnéesNU}^{moy}(R)$	$= \left(\sum_{i=1}^{i \leq Nb_{att}(R)} T_{val}^{moy}(R, att_i) \right) \times T_{caractère}^{fixe}$	
	$= (15 + 75 + 50 + 50 + 4 + 6) \times 2$ $= 400 \text{ octets}$	$= (50 + 300 + 50) \times 2$ $= 800 \text{ octets}$
$T_{métaNU}^{fixe}(R)$	$= T_{drapeau}^{fixe} + (Nb_{att}(R) \times T_{TvalAtt}^{fixe})$	
	$= 1 + (6 \times 2)$ $= 13 \text{ octets}$	$= 1 + (3 \times 2)$ $= 7 \text{ octets}$
$T_{totalNU}^{moy}(R)$	$= T_{donnéesNU}^{moy}(R) + T_{métaNU}^{fixe}(R)$	
	$= 400 + 13$ $= 413 \text{ octets}$	$= 800 + 7$ $= 807 \text{ octets}$

Paramètre calculé (fin)	Relation R	
	LIVRE	EDITEUR
$Nb_{NU/page}^{max}(R)$	$\frac{\left(T_{page}^{fixe} - (Nb_{casesRep}^{fixe} \times T_{caseRep}^{fixe}) \right)}{T_{totalNU}^{moy}}$	
	= (2 048 - (10 × 2))/413	= (2 048 - (10 × 2))/807
	= 4 n-uplets par page	= 2 n-uplets par page
<i>Dans les 2 cas (LIVRE et EDITEUR), la taille fixe du répertoire des déplacements (géré statiquement) ne contraint pas ce nombre de n-uplets par page (le nombre calculé ne dépassant pas le nombre fixe de cases dans le répertoire des déplacements).</i>		
$Nb_{pages}^{min}(R)$	$\frac{Card(R)}{Nb_{NU/page}^{max}(R)}$	
	= 12 000/4 = 3 000 pages	= 300/2 = 150 pages

- Pour les paramètres calculés liés aux index :



Remarque

Contrairement aux paramètres calculés liés aux relations (toujours utiles si les paramètres concernés ne sont pas donnés), les paramètres calculés liés aux index ne sont utiles que pour le calcul des projections (sans élimination des doublons) avec un parcours indexé (cf. §9.1.3.3.2). Ainsi, si aucune des projections au sein d'un arbre de requête ne peut se faire avec cette méthode de résolution, le calcul des paramètres indiqués ci-dessous ne sert à rien : il n'est donc pas forcément utile de les faire dès le départ...

Paramètre calculé (début)	Index I (R, c)				
	Défini sur LIVRE			Défini sur EDITEUR	
	ISBN	NomEdit	Année	Nom	Pays
$Nb_{NU/EI}^{moy}(I)$	$\frac{Card(R)}{Nb_{valDiff}(R, c)}$				
	= 12 000/12 000 = 1 NU/EI	= 12 000/300 = 40 NU/EI	= 12 000/200 = 60 NU/EI	= 300/300 = 1 NU/EI	= 300/10 = 30 NU/EI
$T_{EI}^{moy}(I)$	$\frac{\text{valeur du constituant indexé}}{(T_{val}^{moy}(R, c) \times T_{caractère}^{fixe})} + \frac{\text{adresses logiques associées}}{(Nb_{NU/EI}^{moy}(I) \times T_{adrLog}^{fixe})}$				
	= (15 × 2) + (1 × (4 + 1)) = 35 octets	= (50 × 2) + (40 × (4 + 1)) = 300 octets	= (4 × 2) + (60 × (4 + 1)) = 308 octets	= (50 × 2) + (1 × (4 + 1)) = 105 octets	= (50 × 2) + (30 × (4 + 1)) = 250 octets

Paramètre calculé (fin)	Index I (R, c)				
	Défini sur LIVRE			Défini sur EDITEUR	
	<u>ISBN</u>	NomEdit	Année	<u>Nom</u>	Pays
$\frac{(T_{page}^{fixe} - T_{idPage}^{fixe})}{T_{EI}^{moy}(I)}$					
$Nb_{EI/page}^{moy}(I)$	= (2 048 – 4)/35 = 58 EI/page	= (2 048 – 4)/300 = 6 EI/page	= (2 048 – 4)/308 = 6 EI/page	= (2 048 – 4)/105 = 19 EI/page	= (2 048 – 4)/250 = 8 EI/page
$Nb_{EI}(I)$	$= Nb_{valDiff}(R, c)$				
	= 12 000 EI	= 300 EI	= 200 EI	= 300 EI	= 10 EI
$Nb_{pagesEI}^{min}(I)$	$\frac{Nb_{EI}(I)}{Nb_{EI/page}^{moy}(I)}$				
	= (12 000)/58 = 207 pages	= 300/6 = 50 pages	= 200/6 = 34 pages	= 300/19 = 16 pages	= 10/8 = 2 pages

- Pour les paramètres liés à la requête :

Paramètre calculé	Facteur de sélectivité de la condition	
Condition	Théorique	Valeur
Sélection S_1 (e.Pays = 'Espagne')	$F_{sel}(EDITEUR e, e.Pays = 'Espagne') = \frac{1}{Nb_{valDiff}(EDITEUR, Pays)}$	= 1/10
Sélection S_2 (l.Année = 2000)	$F_{sel}(LIVRE l, l.Année = 200) = \frac{1}{Nb_{valDiff}(LIVRE, Année)}$	= 1/200
Jointure J (e.Nom = l.NomEdit)	$F_{join}(LIVRE l, EDITEUR e, e.Nom = l.NomEdit) = \frac{1}{\max(Nb_{valDiff}(LIVRE, NomEdit), Nb_{valDiff}(EDITEUR, Nom))}$	= 1/300

9.1.5.3 Arbre de requête initial : élaboration et étiquetage

Pour la construction de l'arbre de requête initial, on construit forcément un arbre de requête de la forme jointures/sélection/projection. De plus, on doit suivre les 2 contraintes suivantes (qui sont imposées par les paramètres données) :

- Ce sera un arbre linéaire gauche de jointure (ALGJ),
- Les relations sont introduites (dans la mesure de possible, i.e. en respectant les jointures apparaissant dans la requête) par ordre croissant de leur cardinalité.

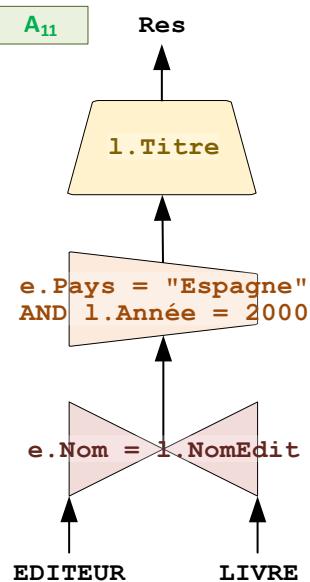


Figure 151. Optimisation (étude de cas) : arbre de requête initial A_{11}

9.1.5.4 Construction (partielle) d'arbres de requête équivalents

L'application des transformations d'arbres T_1 à T_6 permet d'obtenir des arbres équivalents à l'arbre de requête initial. En voici une partie :

- L'arbre A_{12} résulte simplement de l'application de la transformation T_5 (commutativité de la jointure) sur l'arbre de requête initial A_{11} :

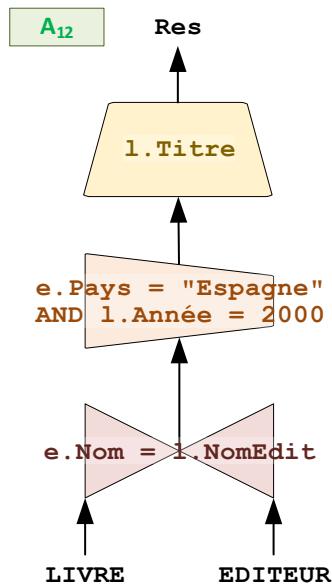


Figure 152. Optimisation (étude de cas) : arbre de requête équivalent A_{12}

- L'arbre A_{21} résulte de l'application à l'arbre de requête initial A_{11} de la transformation T_1 (regroupement/éclatement/permutation des sélections) puis de la transformation T_3 (permutation sélection \leftrightarrow projection) pour descendre sous la jointure la sélection portant sur la relation externe (EDITEUR) ; l'arbre A_{22} résulte du même traitement appliqué à l'arbre A_{12} :

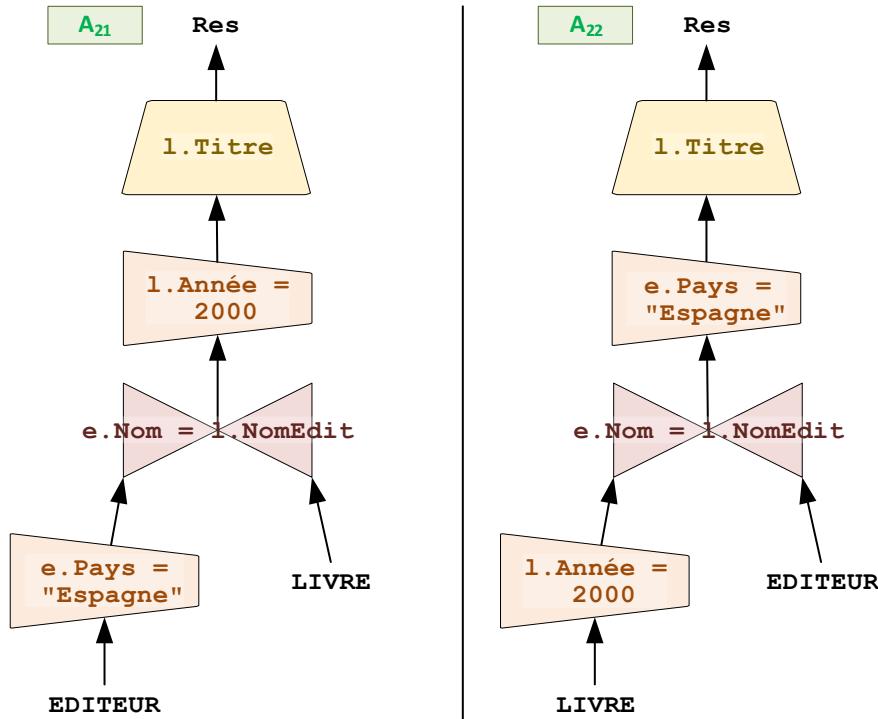


Figure 153. Optimisation (étude de cas) : arbres de requête équivalents A_{21} et A_{22}

- L’arbre A_{31} résulte de l’application à l’arbre de requête initial A_{11} de la transformation T_1 (regroupement/éclatement/permutation des sélections) puis de la transformation T_3 (permutation sélection \leftrightarrow projection) pour descendre sous la jointure la sélection portant sur la relation interne (LIVRE) ; l’arbre A_{32} résulte du même traitement appliqué à l’arbre A_{12} :

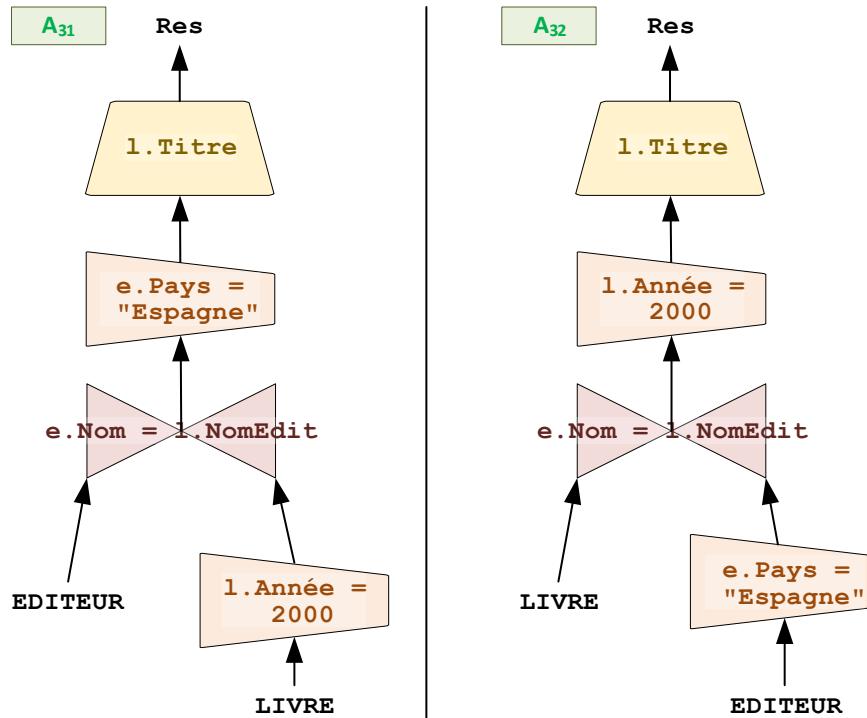


Figure 154. Optimisation (étude de cas) : arbres de requête équivalents A_{31} et A_{32}

- L'arbre A_{41} résulte de l'application à l'arbre de requête initial A_{11} de la transformation T_1 (regroupement/éclatement/permotion des sélections) puis de la transformation T_3 (permotion sélection \leftrightarrow projection) deux fois, pour descendre sous la jointure les 2 sélections ; l'arbre A_{42} résulte du même traitement appliquée à l'arbre A_{12} :

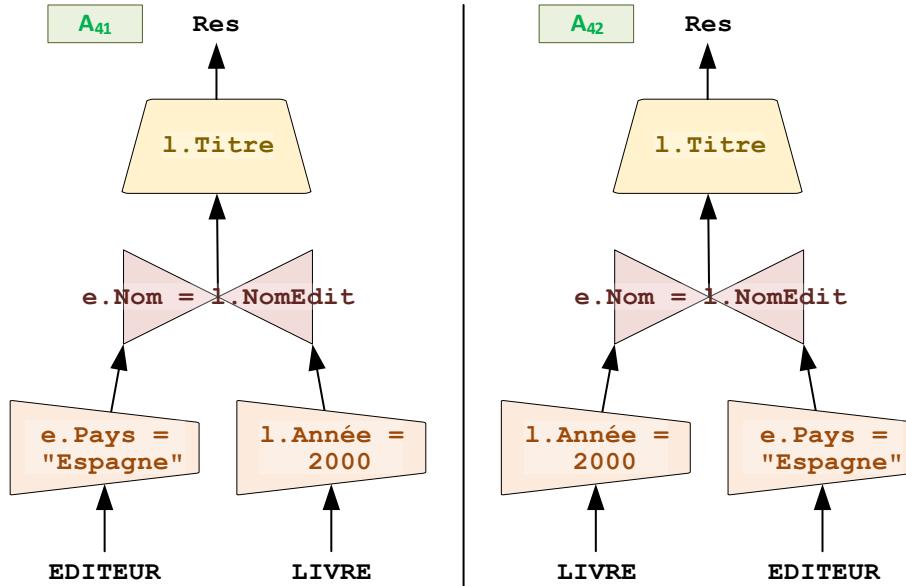


Figure 155. Optimisation (étude de cas) : arbres de requête équivalents A_{41} et A_{42}

On pourrait encore construire d'autres d'arbres de requêtes équivalents : la transformation T_6 (associativité de la jointure) ne peut pas être mise en œuvre (il faut au moins 2 jointures pour cela) mais on pourrait appliquer les transformations T_2 (permotion sélection \leftrightarrow projection) et T_4 (permotion projection \leftrightarrow jointure) sur chacun des 8 arbres précédents pour faire descendre la projection (cette descente devant s'envisager étape par étape, au milieu de l'arbre, jusqu'en bas, ...). Nous nous cantonnerons cependant ici aux 8 arbres de requêtes précédents...

9.1.5.5 Étude (partielle) de coûts : étiquetage des relations et des opérateurs

L'étude des coûts associés aux arbres de requête passe par l'établissement des plans d'exécution que l'on peut leur associer. Cette étude peut néanmoins être réalisée en 1 seule passe : grâce à une technique d'étiquetage, il est ainsi possible de visualiser d'un coup les différents plans d'exécution que l'on peut associer à un arbre de requête et, surtout, leur coût (et, donc, le coût du moins cher de ces plans d'exécution) ! L'idée est donc d'étiqueter les branches et les nœuds des arbres de requête.

9.1.5.5.1 Étiquetage des relations

Les branches (qui représentent les relations réelles, temporaires ou résultat « transitant » dans l'arbre) sont étiquetées par un triplet. Pour chaque relation R (*i.e.* pour chaque branche), il convient ainsi d'indiquer le triplet suivant afin de la « dimensionner » :

$$(Card(R), T_{\text{donnéesNU}}^{\text{fixe|moy|min|max}}(R), Nb_{\text{pages}}^{\text{fixe|moy|min|max}}(R))$$

Équation 65. Triplet étiquetant chaque relation R d'un arbre de requête



En pratique

Ces données sont normalement déjà connues pour les relations réelles (les feuilles de l'arbre) parce qu'elles ont été données (cf. §9.1.5.1) ou déjà calculées (cf. §9.1.5.2). Elles sont en revanche à calculer pour les relations temporaires et la relation résultat de chaque arbre.



Astuces

Fort logiquement, le triplet caractérisant une relation réelle R est identique d'un arbre à un autre arbre équivalent pour cette relation R . Il est donc inutile de le recalculer pour chaque apparition de la relation R d'un arbre de requête à l'autre.

Tout aussi logiquement (mais on y pense parfois moins naturellement), le triplet caractérisant la relation résultat doit être le même d'un arbre à un autre arbre équivalent (puisque, précisément, ils sont équivalents !). Il est cependant conseillé de le recalculer pour chaque arbre : cela peut permettre de vérifier ce que l'on a fait (si le triplet caractérisant la relation résultat diffère d'un arbre à un autre arbre équivalent, c'est qu'il y a un souci quelque part)...

9.1.5.5.2 Étiquetage des opérateurs relationnels

L'étiquetage des nœuds a pour but, lui, d'indiquer les différents coûts de production qu'on peut associer à l'opération relationnel de ce nœud ainsi que son coût d'écriture.



Rappel

On peut associer à un opérateur relationnel plusieurs coûts de production puisque le calcul de ce coût dépend de la méthode de résolution choisie. A contrario, un opérateur relationnel n'a qu'un coût d'écriture, celui-ci ne dépendant que de la relation produite (donc ne dépendant pas de la méthode de résolution choisie).

- Pour un opérateur de sélection : le coût de production présente les 2 alternatives théoriquement possibles (le parcours séquentiel et le parcours indexé) et indique, pour chacune d'entre elles, son coût (indiquer N/A si la méthode de résolution n'a pas les prérequis nécessaires pour être mise en œuvre).

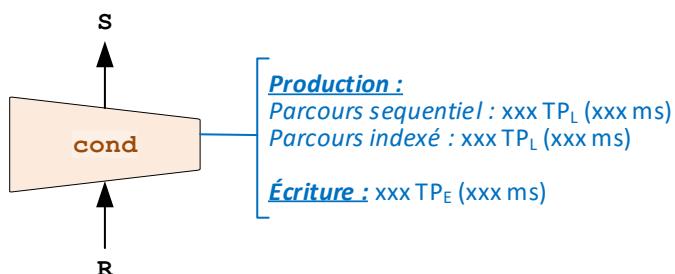


Figure 156. Étiquetage d'une sélection

- Pour un opérateur de jointure : le coût de production présente les 3 alternatives théoriquement possibles (les boucles imbriquées, les boucles optimisées et la boucle indexée) et indique, pour chacune d'entre elles, son coût (indiquer N/A si la méthode de résolution n'a pas les prérequis nécessaires pour être mise en œuvre).

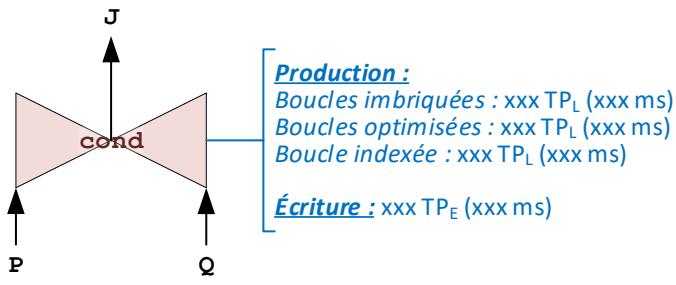


Figure 157. Étiquetage d'une jointure

- Pour un opérateur de projection : le coût de production présente les 2 alternatives théoriquement possibles (le parcours séquentiel et le parcours indexé) et indique, pour chacune d'entre elles, son coût (indiquer N/A si la méthode de résolution n'a pas les prérequis nécessaires pour être mise en œuvre).

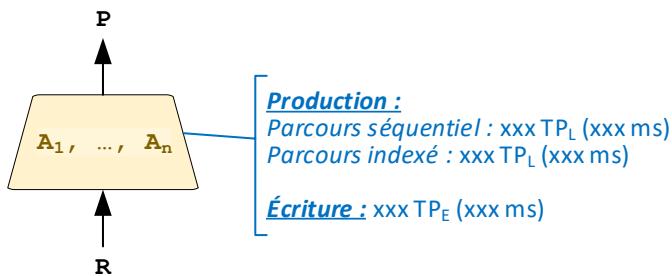


Figure 158. Étiquetage d'une projection

9.1.5.5.3 Étiquetage d'un arbre de requête

Usuellement, un arbre de requête (*i.e.* les relations ET les opérateurs relationnels qui y figurent) peut s'étiqueter en 1 ou 2 passes, qui s'effectuent de bas en haut de la façon suivante (si on étiquette l'arbre en 1 seule passe) pour chaque opérateur relationnel :

- Les relations entrant dans l'opérateur sont étiquetées.



En pratique

Si ces relations sont des relations réelles ou produites par un opérateur relationnel situé plus bas dans l'arbre de requête, elles sont normalement déjà étiquetées !

- Puisqu'on connaît les données d'entrée, on peut calculer les différents coûts de production associés à l'opérateur relationnel considéré.



Remarque

La seule chose qui peut éventuellement manquer c'est le nombre de pages occupées par les entrées d'un index pour la mise en œuvre de la méthode de résolution d'une projection par parcours indexé.

3. La relation produite par l'opérateur est étiquetée.



En pratique

Cette étape peut même être réalisée avant l'étape 2 : il est ainsi possible, pour un arbre de requête donné, d'étiqueter d'un coup toutes les relations (de bas en haut) puis tous les opérateurs relationnels (de bas en haut de nouveau).

4. Le coût d'écriture de l'opérateur relationnel considéré est calculé (puisque il est égal au nombre de pages de la relation produite).



En pratique

Indiquer le coût d'écriture d'un opérateur relationnel consiste donc juste à reporter ici la dernière valeur du triplet caractérisant la relation produite par cet opérateur.

9.1.5.5.4 Exemple d'étiquetage d'arbres de requête équivalents

En considérant les 8 arbres de requêtes équivalents A_{11} à A_{42} , on doit maintenant réaliser leur étiquetage.



Remarque

Nous n'allons ici le faire QUE pour les arbres de requête A_{x1} , i.e. ceux dans lesquels la relation externe de la jointure est issue (directement ou indirectement) de la relation EDITEUR. Nous avons choisi de nous focaliser sur ceux-là précisément parce que la relation EDITEUR est celle de plus petite cardinalité ET occupant le moins de pages (et nous avons vu, lors de l'étude des coûts des méthodes de résolution de la jointure, qu'il valait mieux, selon la méthode, que la relation externe soit celle qui occupe le moins de pages, cf. §9.1.3.2.1 et §9.1.3.2.2, ou celle de plus petite cardinalité, cf. §9.1.3.2.3).

Néanmoins, dans la pratique, il faudrait tout de même réaliser cet étiquetage pour les arbres de requête A_{x2} , ainsi que pour les autres arbres de requête équivalents que nous n'avons pas construits !

9.1.5.5.4.1 Étiquetage de l'arbre de requête initial A_{11}

Concernant les relations réelles EDITEUR et LIVRE, nous avons déjà les données permettant de les étiqueter :

Paramètre	Relation R	
	LIVRE	EDITEUR
$Card(R)$	12 000 n-uplets	300 n-uplets
$T_{donnéesNU}^{moy}(R)$	400 octets	800 octets
$Nb_{pages}^{min}(R)$	3 000 pages	150 pages

Ce même travail est à réaliser pour la relation temporaire issue de la jointure (appelons-la Tmp_1), pour la relation temporaire issue de la sélection (appelons-la Tmp_2) et pour la relation résultat (Res).

Paramètre calculé	Relation R		
	Tmp_1	Tmp_2	Res
R produite par	Jointure J	Sélection S	Projection P
$\text{Card}(\text{R})$	$= \text{Card}(EDITEUR) \times \text{Card}(LIVRE) \times F_{join}(J)$ $= 300 \times 12\,000 \times (1/300)$ $= 12\,000 \text{ n-uplets}$	$= \text{Card}(\text{Tmp}_1) \times F_{sel}(S \equiv S_1 \wedge S_2)$ $= 12\,000 \times (1/10 \times 1/200)$ $= 6 \text{ n-uplets}$	$= \text{Card}(\text{Tmp}_2)$ $= 6 \text{ n-uplets}$
$T_{\text{donnéesNU}}^{\text{moy}}(\text{R})$	$= T_{\text{donnéesNU}}^{\text{moy}}(EDITEUR) + T_{\text{donnéesNU}}^{\text{moy}}(LIVRE)$ $= 800 + 400$ $= 1\,200 \text{ octets}$	$= T_{\text{donnéesNU}}^{\text{moy}}(\text{Tmp}_1)$ $= 1\,200 \text{ octets}$	$= T_{\text{val}}^{\text{moy}}(LIVRE, Titre) \times T_{\text{caractère}}^{\text{fixe}}$ $= (75 \times 2)$ $= 150 \text{ octets}$
$T_{\text{métaNU}}^{\text{fixe}}(\text{R})$	$= T_{\text{drapeau}}^{\text{fixe}} + (Nb_{att}(\text{R}) \times T_{\text{valAtt}}^{\text{fixe}})$ $= 1 + (9 \times 2)$ $= 19 \text{ octets}$	$= 1 + (9 \times 2)$ $= 19 \text{ octets}$	$= 1 + (1 \times 2)$ $= 3 \text{ octets}$
$T_{\text{totalNU}}^{\text{moy}}(\text{R})$	$= T_{\text{donnéesNU}}^{\text{moy}}(\text{R}) + T_{\text{métaNU}}^{\text{fixe}}(\text{R})$ $= 1\,200 + 19$ $= 1\,219 \text{ octets}$	$= 1\,200 + 19$ $= 1\,219 \text{ octets}$	$= 150 + 3$ $= 153 \text{ octets}$
$Nb_{\text{NU}/page}^{\text{max}}(\text{R})$	$\frac{(T_{\text{page}}^{\text{fixe}} - (Nb_{casesRep}^{\text{fixe}} \times T_{\text{caseRep}}^{\text{fixe}}))}{T_{\text{totalNU}}^{\text{moy}}}$ $= \frac{(2\,048 - (10 \times 2))/1\,219}{153}$ $= 1 \text{ NU/page}$		
	$Dans les 2 cas (\text{Tmp}_1 \text{ et } \text{Tmp}_2), la taille fixe du répertoire des déplacements (géré statiquement) ne constraint pas ce nombre de n-uplets par page.$		
$Nb_{\text{pages}}^{\text{min}}(\text{R})$	$\frac{\text{Card}(\text{R})}{Nb_{\text{NU}/page}^{\text{max}}(\text{R})}$ $= \frac{12\,000}{1} = 12\,000 \text{ pages}$		
	$= 6/1 = 6 \text{ pages}$		
	$= 6 / 10 = 1 \text{ page}$		

À ce stade, on a donc tout ce qu'il faut pour étiqueter les relations de l'arbre de requête initial A_{11} !

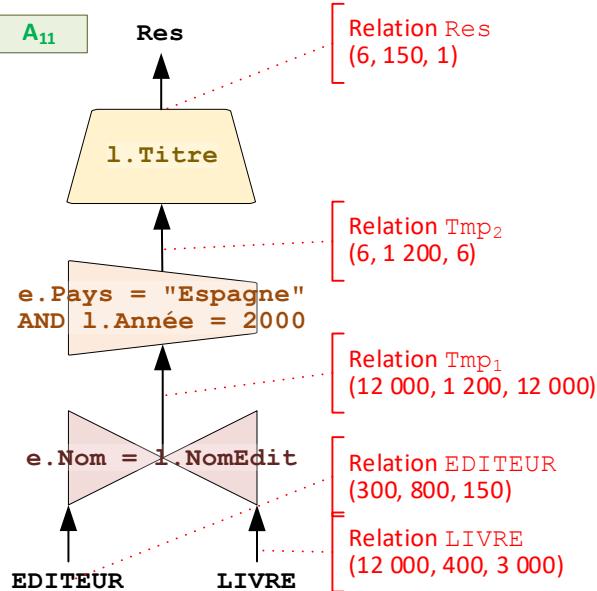


Figure 159. Optimisation (étude de cas) : étiquetage des relations de l'arbre de requête A₁₁

Il est donc maintenant nécessaire d'étiqueter ses nœuds. Les coûts de production associés aux méthodes de résolution de chaque opérateur²¹⁰ et les coûts d'écriture dépendant directement de l'étiquetage des relations



Rappel

Cela aurait aussi pu être réalisé dans la même passe, toujours de bas en haut. Mais nous avons préféré, pour cet exemple détaillé, distinguer l'étiquetage des relations de celui des opérateurs relationnels.



Attention

Pour savoir si une méthode de résolution est applicable (i.e. si ses prérequis sont satisfaits), n'oubliez notamment surtout pas :

- De prendre en compte les pages punaisées en mémoire cache (pour savoir combien de cases restent disponibles en mémoire cache),
- De prendre en compte les possibilités de propagation des index et l'éventuel groupement des index (pour les méthodes de résolution basées sur l'usage d'index).

Dans notre exemple, la mémoire cache peut contenir 50 pages (puisque elle contient 50 cases en tout). Mais, on nous dit aussi que, pour chaque index en arbre B+, la racine de cet arbre est punaisée en mémoire cache. Étant donné que nous avons 5 index en arbre B+, il reste donc 45 cases disponibles en mémoire cache pour la mise en œuvre des méthodes de production des opérateurs relationnels.

²¹⁰ Encore une fois, hormis la méthode de résolution d'une projection par parcours indexé qui nécessite, elle, de connaître le nombre de pages contenant des entrées d'index dans l'index utilisé pour sa mise en œuvre.

Opérateur relationnel (A_{11})		Méthode de résolution	Prérequis ?	Coût de production
Jointure J	Production	Boucles imbriquées	Oui	= 150 + (150 × 3 000) = 450 150 transferts de pages (en lecture) ≈ 450 150 ms
		Boucles optimisées	Oui	= 150 + ((150/43) \uparrow × 3 000) = 12 150 transferts de pages (en lecture) ≈ 12 150 ms
		Boucle indexée	Oui	= 150 + 300 × (2 + (3000/300) \uparrow) = 3 750 transferts de pages (en lecture) ≈ 3 750 ms
Écriture				= 12 000 transferts de pages (en écriture) = 24 000 ms
Sélection S	Production	Parcours séquentiel	Oui	= 12 000 transferts de pages (en lecture) ≈ 12 000 ms
		Parcours indexé	Non	N/A : les index n'étant pas propagés au-delà des opérateurs relationnels, la relation d'entrée de la sélection (Tmp_1) n'est pas indexée.
	Écriture			= 6 transferts de pages (en écriture) ≈ 12 ms
Projection P	Production	Parcours séquentiel	Oui	= 6 transferts de pages (en lecture) ≈ 6 ms
		Parcours indexé	Non	N/A : les index n'étant pas propagés au-delà des opérateurs relationnels, la relation d'entrée de la projection (Tmp_2) n'est pas indexée.
	Écriture			= 1 transfert de page (en écriture) ≈ 2 ms

L'arbre A_{11} une fois étiqueté est le suivant :

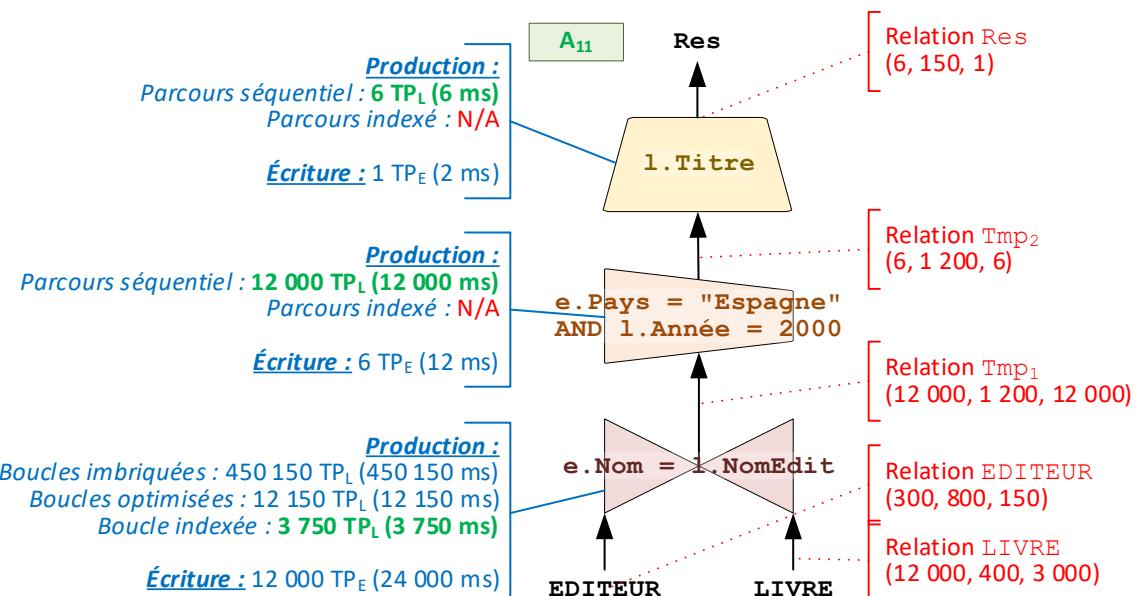


Figure 160. Optimisation (étude de cas) : étiquetage complet de l'arbre de requête A_{11}

Cet étiquetage nous « apprend » les points suivants concernant l’arbre de requête A_{11} :

- En théorie (*i.e.* si toutes les méthodes de résolution de ses opérateurs relationnels étaient applicables), on aurait pu lui associer 12 plans d’exécution (3 méthodes de résolution connues pour la jointure \times 2 méthodes de résolution connues pour la sélection \times 2 méthodes de résolution connues pour la projection),
- En pratique (*i.e.* en tenant cette fois compte de la satisfaction des prérequis des méthodes de résolution de ses opérateurs relationnels), on peut lui associer 3 plans d’exécution (3 méthodes de résolution « activables » pour la jointure \times 1 méthode de résolution « activable » pour la sélection \times 1 méthode de résolution « activable » pour la projection),
- Le moins cher des 3 plans d’exécution effectivement associables à cet arbre de requête coûte : $(3\ 750 + 12\ 000 + 6) \text{ TP}_L + (12\ 000 + 6 + 1) \text{ TP}_E = 15\ 756 \text{ TP}_L + 12\ 007 \text{ TP}_E$ (soit 39 770 ms \approx 40 s).

Le même travail peut être réalisé pour les arbres de requêtes A_{21} , A_{31} et A_{41} ...

9.1.5.5.4.2 Étiquetage des relations des arbres de requête A_{21} , A_{31} et A_{41}

Concernant les relations temporaires et résultat de ces arbres, les données suivantes sont calculées de la même façon que pour l’arbre de requête initial A_{11} :

- Pour l’arbre de requête A_{21} :

Paramètre calculé (A_{21})	Relation R			
	Tmp ₁	Tmp ₂	Tmp ₃	Res
R produite par	Sélection S ₁ sur e.PAYS	Jointure J	Sélection S ₂ sur l.Année	Projection P
Card(R)	= $300 \times (1/10)$ = 30 n-uplets	= $30 \times 12\ 000 \times (1/300)$ = 1 200 n-uplets	= $1\ 200 \times (1/200)$ = 6 n-uplets	= 6 n-uplets
T _{donnéesNU} ^{moy} (R)	= 800 octets	= 800 + 400 = 1 200 octets	= 1 200 octets	= 150 octets
T _{métaNU} ^{fixe} (R)	= $1 + (3 \times 2)$ = 7 octets	= $1 + (9 \times 2)$ = 19 octets	= $1 + (9 \times 2)$ = 19 octets	= $1 + (1 \times 2)$ = 3 octets
T _{totalNU} ^{moy} (R)	= $800 + 7$ = 807 octets	= $1\ 200 + 19$ = 1 219 octets	= $1\ 200 + 19$ = 1 219 octets	= $150 + 3$ = 153 octets
Nb _{NU/page} ^{max} (R)	= $\lceil (2\ 048 - (10 \times 2)) / 807 \rceil$ = 2 NU/page	= $\lceil (2\ 048 - (10 \times 2)) / 1\ 219 \rceil$ = 1 NU/page	= $\lceil (2\ 048 - (10 \times 2)) / 1\ 219 \rceil$ = 1 NU/page	= $\lceil (2\ 048 - (10 \times 2)) / 153 \rceil$ = 13 NU/page ≈ 10 NU/page
Nb _{pages} ^{min} (R)	= $\lceil 30/2 \rceil$ = 15 pages	= $\lceil 1\ 200/1 \rceil$ = 1 200 pages	= $\lceil 6/1 \rceil$ = 6 pages	= $\lceil 6/10 \rceil$ = 1 page

- Pour l'arbre de requête A₃₁ :

Paramètre calculé (A ₃₁)	Relation R			
	Tmp ₁	Tmp ₂	Tmp ₃	Res
R produite par	Sélection S ₁ sur l . Année	Jointure J	Sélection S ₂ sur e . Pays	Projection P
Card(R)	= 12 000 × (1/200) = 60 n-uplets	= 300 × 60 × (1/300) = 60 n-uplets	= 60 × (1/10) = 6 n-uplets	= 6 n-uplets
T ^{moy} _{donnéesNU(R)}	= 400 octets	= 800 + 400 = 1 200 octets	= 1 200 octets	= 150 octets
T ^{fixe} _{métaNU(R)}	= 1 + (6 × 2) = 13 octets	= 1 + (9 × 2) = 19 octets	= 1 + (9 × 2) = 19 octets	= 1 + (1 × 2) = 3 octets
T ^{moy} _{totalNU(R)}	= 400 + 13 = 413 octets	= 1 200 + 19 = 1 219 octets	= 1 200 + 19 = 1 219 octets	= 150 + 3 = 153 octets
Nb ^{max} _{NU/page(R)}	= ↘ (2 048 – (10 × 2))/413 = 4 NU/page	= ↘ (2 048 – (10 × 2))/1 219 = 1 NU/page	= ↘ (2 048 – (10 × 2))/1 219 = 1 NU/page	= ↘ (2 048 – (10 × 2))/153 = 13 NU/page ≈ 10 NU/page
Nb ^{min} _{pages(R)}	= ↗ 60/4 = 15 pages	= ↗ 60/1 = 60 pages	= ↗ 6/1 = 6 pages	= ↗ 6/10 = 1 page

- Pour l'arbre de requête A₄₁ :

Paramètre calculé (A ₄₁)	Relation R			
	Tmp ₁	Tmp ₂	Tmp ₃	Res
R produite par	Sélection S ₁ sur e . Pays	Sélection S ₂ sur l . Année	Jointure J	Projection P
Card(R)	= 300 × (1/10) = 30 n-uplets	= 12 000 × (1/200) = 60 n-uplets	= 30 × 60 × (1/300) = 6 n-uplets	= 6 n-uplets
T ^{moy} _{donnéesNU(R)}	= 800 octets	= 400 octets	= 800 + 400 = 1 200 octets	= 150 octets
T ^{fixe} _{métaNU(R)}	= 1 + (3 × 2) = 7 octets	= 1 + (6 × 2) = 13 octets	= 1 + (9 × 2) = 19 octets	= 1 + (1 × 2) = 3 octets
T ^{moy} _{totalNU(R)}	= 800 + 7 = 807 octets	= 400 + 13 = 413 octets	= 1 200 + 19 = 1 219 octets	= 150 + 3 = 153 octets
Nb ^{max} _{NU/page(R)}	= ↘ (2 048 – (10 × 2))/807 = 2 NU/page	= ↘ (2 048 – (10 × 2))/413 = 4 NU/page	= ↘ (2 048 – (10 × 2))/1 219 = 1 NU/page	= ↘ (2 048 – (10 × 2))/153 = 13 NU/page ≈ 10 NU/page
Nb ^{min} _{pages(R)}	= ↗ 30/2 = 15 pages	= ↗ 60/4 = 15 pages	= ↗ 6/1 = 6 pages	= ↗ 6/10 = 1 page

9.1.5.5.4.3 Étiquetage des opérateurs relationnels des arbres de requête A₂₁, A₃₁ et A₄₁

L'étiquetage des relations étant réalisé pour ces 3 arbres de requête, il est possible de calculer les coûts (de production, selon la méthode de résolution considérée, et d'écriture) pour les opérateurs relationnels figurant en leur sein :

- Pour l'arbre de requête A₂₁ :

Opérateur relationnel (A ₂₁)		Méthode de résolution	Prérequis ?	Coût de production
Sélection S ₁ sur e . Pays	Prod.	Parcours séquentiel	Oui	= 150 transferts de pages (en lecture) ≈ 150 ms
		Parcours indexé	Oui	= 1 + (150/10) [↗] = 16 transferts de pages (en lecture) ≈ 16 ms
	Écriture			= 15 transferts de pages (en écriture) ≈ 30 ms
Jointure J	Prod.	Boucles imbriquées	Oui	= 15 + (15 × 3 000) = 45 015 transferts de pages (en lecture) ≈ 45 015 ms
		Boucles optimisées	Oui	= 15 + ((15/43) [↗] × 3 000) = 3 015 transferts de pages (en lecture) ≈ 3 015 ms
		Boucle indexée	Oui	= 15 + 30 × (2 + (3 000/300) [↗]) = 375 transferts de pages (en lecture) ≈ 375 ms
	Écriture			= 1 200 transferts de pages (en écriture) ≈ 2 400 ms
Sélection S ₂ sur l . Année	Prod.	Parcours séquentiel	Oui	= 1 200 transferts de pages (en lecture) ≈ 1 200 ms
		Parcours indexé	Non	N/A : les index n'étant pas propagés au-delà des opérateurs relationnels, la relation d'entrée de la sélection (Tmp ₂) n'est pas indexée.
	Écriture			= 6 transferts de pages (en écriture) ≈ 12 ms
Projection P	Prod.	Parcours séquentiel	Oui	= 6 transferts de pages en lecture ≈ 6 ms
		Parcours indexé	Non	N/A : les index n'étant pas propagés au-delà des opérateurs relationnels, la relation d'entrée de la projection (Tmp ₃) n'est pas indexée.
	Écriture			= 1 transfert de page (en écriture) ≈ 2 ms

- Pour l'arbre de requête A₃₁ :

Opérateur relationnel (A ₃₁)		Méthode de résolution	Prérequis ?	Coût de production
Sélection S ₁ sur l . Année	Prod.	Parcours séquentiel	Oui	= 3 000 transferts de pages (en lecture) ≈ 3 000 ms
		Parcours indexé	Oui	= 1 + (12 000/200) [↗] = 61 transferts de pages (en lecture) ≈ 61 ms
	Écriture			= 15 transferts de pages (en écriture) ≈ 30 ms
Jointure J	Prod.	Boucles imbriquées	Oui	= 150 + (150 × 15) = 2 400 transferts de pages (en lecture) ≈ 2 400 ms
		Boucles optimisées	Oui	= 150 + ((150/43) [↗] × 15) = 210 transferts de pages (en lecture) ≈ 210 ms
		Boucle indexée	Non	N/A : les index n'étant pas propagés au-delà des opérateurs relationnels, la relation interne de la jointure (T _{mp1}) n'est pas indexée.
	Écriture			= 60 transferts de pages (en écriture) ≈ 120 ms
Sélection S ₂ sur e . Pays	Prod.	Parcours séquentiel	Oui	= 60 transferts de pages (en lecture) ≈ 60 ms
		Parcours indexé	Non	N/A : les index n'étant pas propagés au-delà des opérateurs relationnels, la relation d'entrée de la sélection (T _{mp2}) n'est pas indexée.
	Écriture			= 6 transferts de pages (en écriture) ≈ 12 ms
Projection P	Prod.	Parcours séquentiel	Oui	= 6 transferts de pages (en lecture) ≈ 6 ms
		Parcours indexé	Non	N/A : les index n'étant pas propagés au-delà des opérateurs relationnels, la relation d'entrée de la projection (T _{mp3}) n'est pas indexée.
	Écriture			= 1 transfert de page (en écriture) ≈ 2 ms

- Enfin, pour l'arbre de requête A₄₁ :

Opérateur relationnel (A ₄₁)		Méthode de résolution	Prérequis ?	Coût de production
Sélection S ₁ sur e . Pays	Prod.	Parcours séquentiel	Oui	= 150 transferts de pages (en lecture) ≈ 150 ms
		Parcours indexé	Oui	= 1 + (150/10) \nearrow = 16 transferts de pages (en lecture) ≈ 16 ms
	Écriture			= 15 transferts de pages (en écriture) ≈ 30 ms
Sélection S ₂ sur l . Année	Prod.	Parcours séquentiel	Oui	= 3 000 transferts de pages (en lecture) ≈ 3 000 ms
		Parcours indexé	Oui	= 1 + (12 000/200) \nearrow = 61 transferts de pages (en lecture) ≈ 61 ms
	Écriture			= 15 transferts de pages (en écriture) ≈ 30 ms
Jointure J	Prod.	Boucles imbriquées	Oui	= 15 + (15 × 15) = 240 transferts de pages (en lecture) ≈ 240 ms
		Boucles optimisées	Oui	= 15 + ((15/43) \nearrow × 15) = 30 transferts de pages (en lecture) ≈ 30 ms
		Boucle indexée	Non	N/A : les index n'étant pas propagés au-delà des opérateurs relationnels, la relation interne de la jointure (Tmp ₂) n'est pas indexée.
	Écriture			= 6 transferts de pages (en écriture) ≈ 12 ms
Projection P	Prod.	Parcours séquentiel	Oui	= 6 transferts de pages (en lecture) ≈ 6 ms
		Parcours indexé	Non	N/A : les index n'étant pas propagés au-delà des opérateurs relationnels, la relation d'entrée de la projection (Tmp ₃) n'est pas indexée.
	Écriture			= 1 transfert de page (en écriture) ≈ 2 ms

Globalement, complètement étiquetés, les arbres de requête A_{21} , A_{31} et A_{41} sont les suivants :

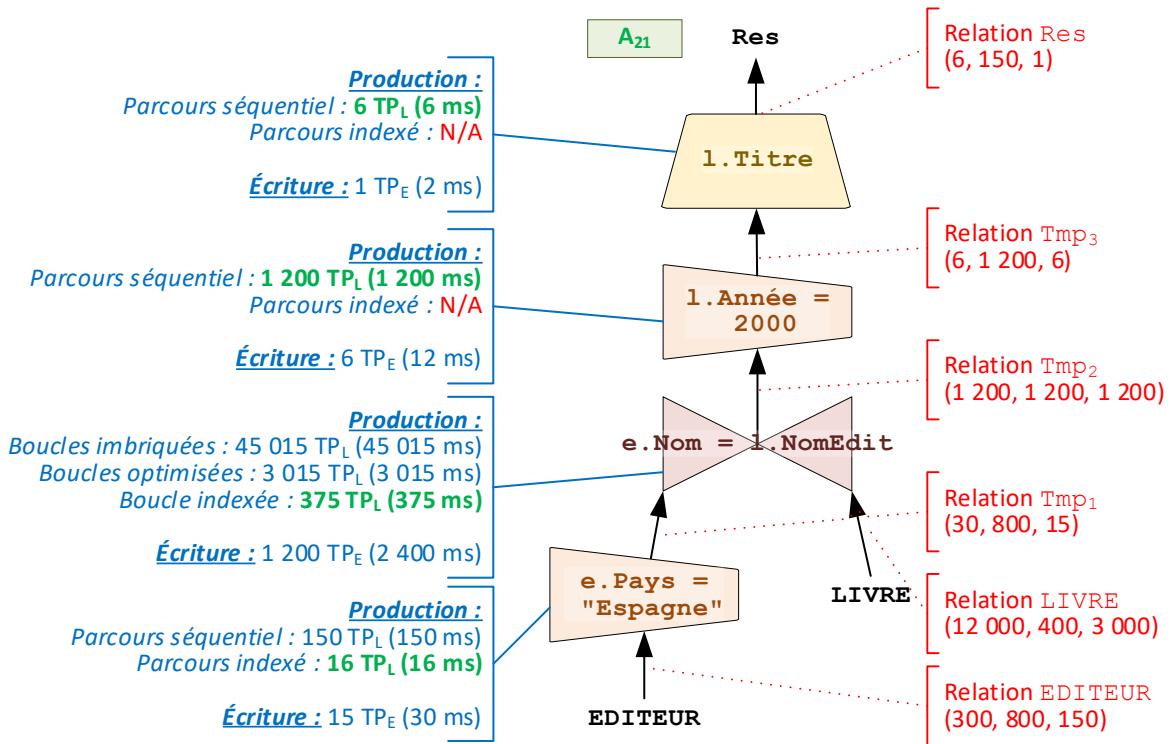


Figure 161. Optimisation (étude de cas) : étiquetage complet de l'arbre de requête A_{21}

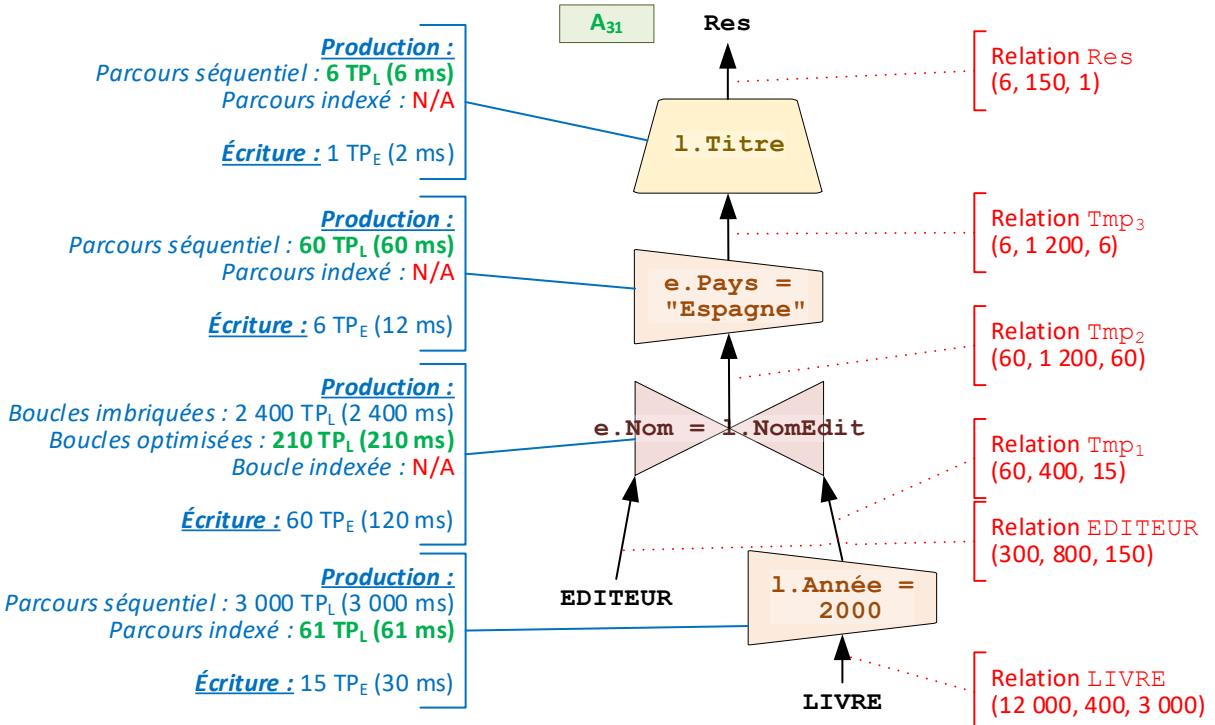


Figure 162. Optimisation (étude de cas) : étiquetage complet de l'arbre de requête A_{31}

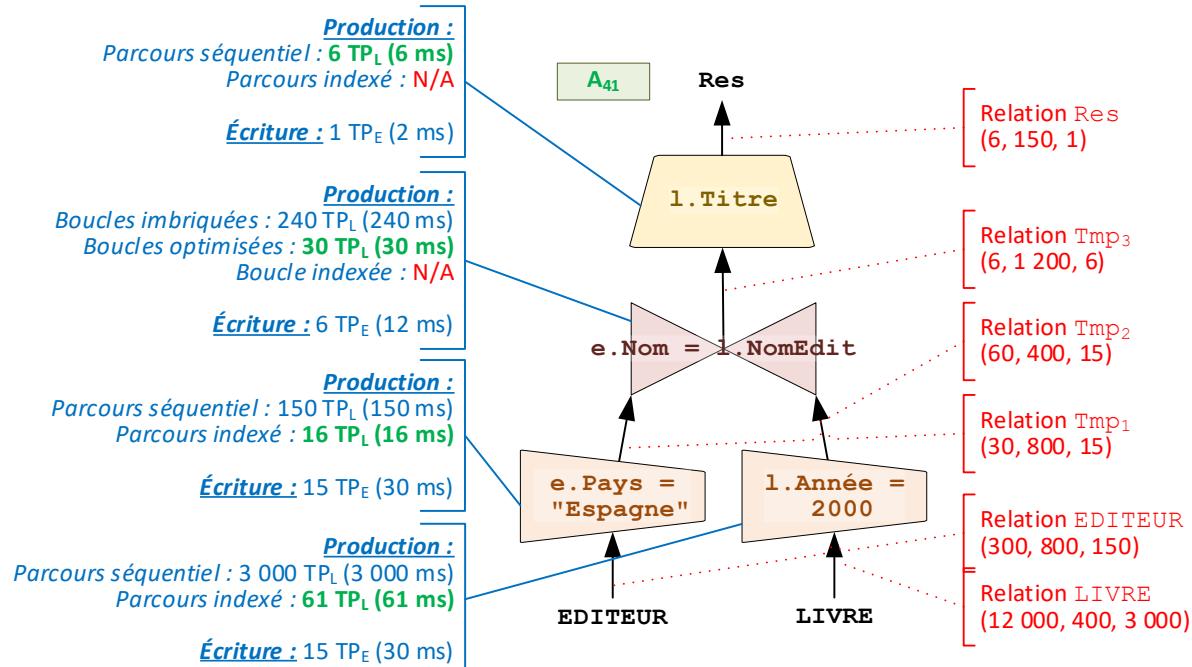


Figure 163. Optimisation (étude de cas) : étiquetage complet de l'arbre de requête A₄₁

9.1.5.5.5 Conclusion de l'étude (partielle)

Finalement, pour les 4 arbres de requêtes que nous avons considérés, nous avons :

Arbre de requête	Nombre de plans d'exécution associables		Coût du moins cher des plans d'exécution effectivement associables à l'arbre de requête	
	Théorique	Effectif	Transferts de pages	Temps
A ₁₁	12	3	15 756 TP _L + 12 007 TP _E	39 770 ms ≈ 40 s
A ₂₁	24	6	1 597 TP _L + 1 222 TP _E	4 041 ms ≈ 4 s
A ₃₁	24	4	337 TP _L + 80 TP _E	497 ms ≈ 0,5 s
A ₄₁	24	8	113 TP _L + 37 TP _E	187 ms ≈ 0,2 s

Parmi ces 4 arbres de requête équivalents, c'est l'arbre A₄₁ qui est associé au plan d'exécution le plus performant, ce dernier ayant un coût global de 187 ms.



Remarque

Ce ne sera peut-être pas ce plan d'exécution qui sera mis en œuvre : n'oubliez pas que nous n'avons pas étudié (donc pas pris en compte) tous les autres arbres de requête équivalents (et les plans d'exécution associés).



Attention

Les résultats auraient pu être complètement différents selon le paramétrage du SGBD, notamment selon les possibilités de propagation des index.

Le fait que ce soit l'arbre A_{41} qui soit associé au plan d'exécution le plus performant n'est finalement pas étonnant : c'est celui qui réduit le plus possible la taille des relations entrant dans la jointure (qui est l'opérateur le plus coûteux).

On peut essayer d'améliorer encore le résultat en descendant également la projection entre les sélections et la jointure (c'est ce qui est fait dans l'arbre A_{51} ci-dessous, obtenu en descendant la projection sous la jointure mais au-dessus des sélections²¹¹ à l'aide de la transformation T_4 de permutation projection \leftrightarrow jointure appliquée à l'arbre de requête A_{41}) :

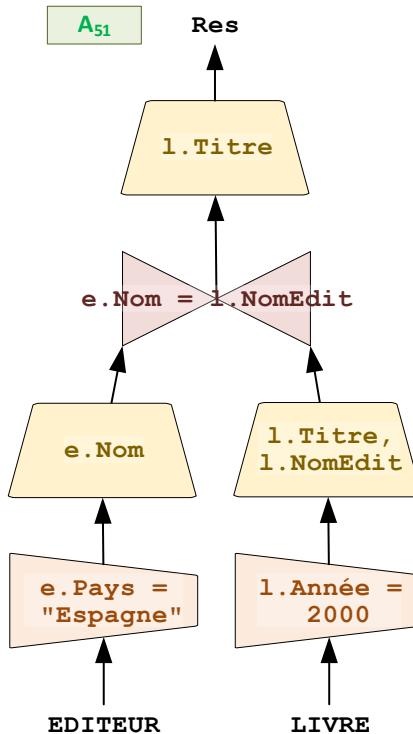


Figure 164. Optimisation (étude de cas) : application de l'heuristique DSP à l'arbre de requête A_{41}

Dès lors, on peut réaliser l'étiquetage de cet arbre de requête, à commencer par celui des relations...

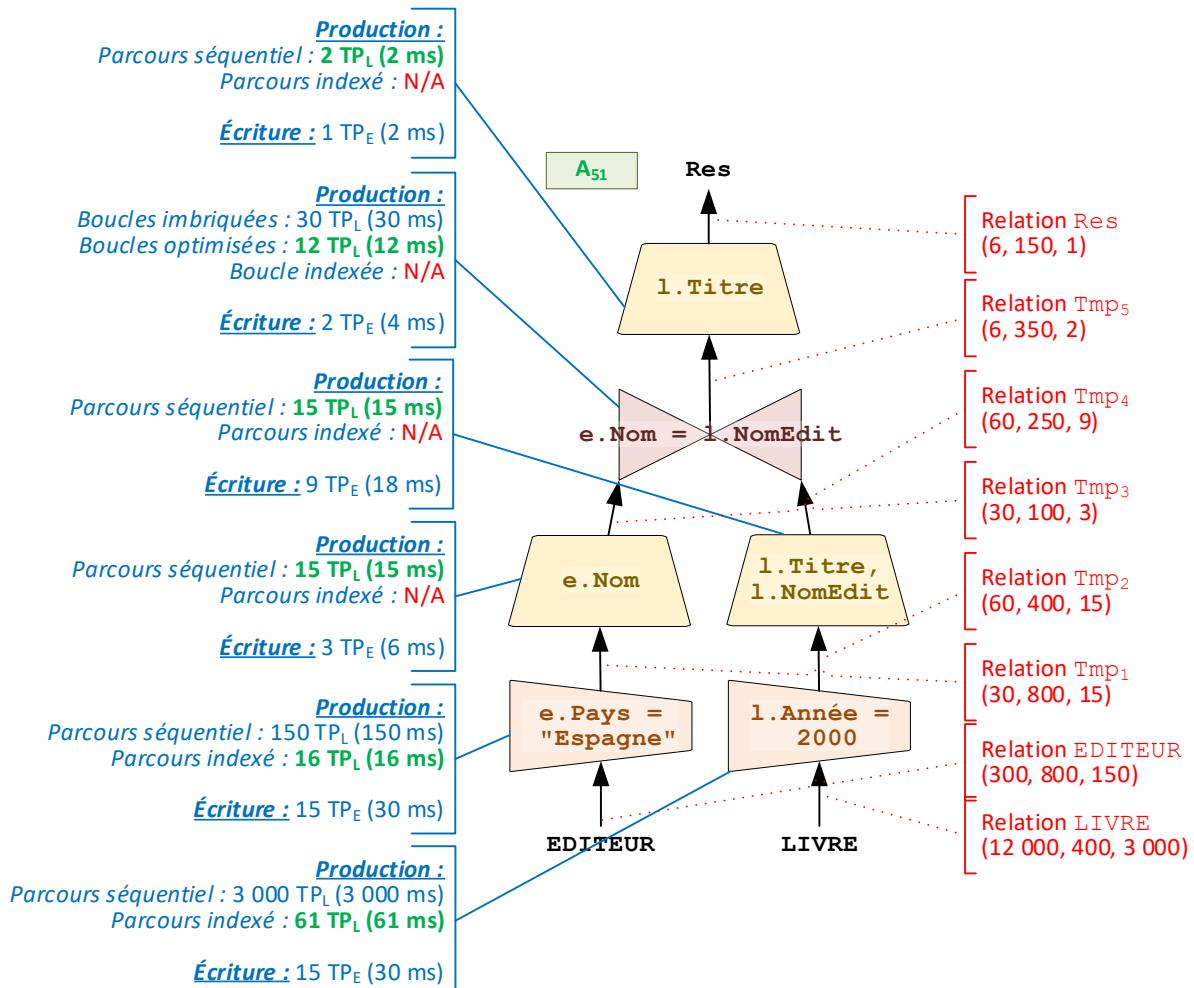
²¹¹ Comme le ferait l'heuristique de descente des sélections puis des projections (cf. §9.1.1.3.2).

Paramètre calculé (A_{51})	Relation R					
	Tmp ₁	Tmp ₂	Tmp ₃	Tmp ₄	Tmp ₅	Res
R produite par	Sélection S ₁ sur e.Pays	Sélection S ₂ sur l.Année	Projection P ₁ sur e.Nom	Projection P ₂ sur {l.Titre, l.NomEdit}	Jointure J	Projection P ₃ sur l.Titre
Card(R)	= 300 × (1/10) = 30 NU	= 12 000 × (1/200) = 60 NU	= 30 NU	= 60 NU	= 30 × 60 × (1/300) = 6 NU	= 6 NU
T ^{moy} _{donnéesNU} (R)	= 800 octets	= 400 octets	= (50 × 2) = 100 octets	= (75 + 50) × 2 = 250 octets	= 100 + 250 = 350 octets	= 150 octets
T ^{fixe} _{métaNU} (R)	= 1 + (3 × 2) = 7 octets	= 1 + (6 × 2) = 13 octets	= 1 + (1 × 2) = 3 octets	= 1 + (2 × 2) = 5 octets	= 1 + (3 × 2) = 7 octets	= 1 + (1 × 2) = 3 octets
T ^{moy} _{totalNU} (R)	= 800 + 7 = 807 octets	= 400 + 13 = 413 octets	= 100 + 3 = 103 octets	= 250 + 5 = 255 octets	= 350 + 7 = 357 octets	= 150 + 3 = 153 octets
Nb ^{max} _{NU/page} (R)	= ↗ (2 048 – (10 × 2))/807 = 2 NU/page	= ↗ (2 048 – (10 × 2))/413 = 4 NU/page	= ↗ (2 048 – (10 × 2))/103 = 19 NU/page ≈ 10 NU/page	= ↗ (2 048 – (10 × 2))/255 = 7 NU/page	= ↗ (2 048 – (10 × 2))/357 = 5 NU/page	= ↗ (2 048 – (10 × 2))/153 = 13 NU/page ≈ 10 NU/page
Nb ^{min} _{pages} (R)	= ↗ 30/2 = 15 pages	= ↗ 60/4 = 15 pages	= ↗ 30/10 = 3 pages	= ↗ 60/7 = 9 pages	= ↗ 6/5 = 2 pages	= ↗ 6/10 = 1 page

D'où l'étiquetage suivant pour les opérateurs relationnels de l'arbre de requête A₅₁...

Opérateur relationnel (A_{51})		Méthode de résolution	Prérequis ?	Coût de production
Sélection S_1 sur $e.Pays$	Prod.	Parcours séquentiel	Oui	= 150 transferts de pages (en lecture) ≈ 150 ms
		Parcours indexé	Oui	= $1 + (150/10)^\wedge$ = 16 transferts de pages (en lecture) ≈ 16 ms
		Écriture		= 15 transferts de pages (en écriture) ≈ 30 ms
Sélection S_2 sur $l.Année$	Prod.	Parcours séquentiel	Oui	= 3 000 transferts de pages (en lecture) $\approx 3 000$ ms
		Parcours indexé	Oui	= $1 + (12\ 000/200)^\wedge$ = 61 transferts de pages (en lecture) ≈ 61 ms
		Écriture		= 15 transferts de pages (en écriture) ≈ 30 ms
Projection P_1 sur $e.Pays$	Prod.	Parcours séquentiel	Oui	= 15 transferts de pages (en lecture) ≈ 15 ms
		Parcours indexé	Non	N/A : la relation d'entrée de la projection (Tmp_1) n'est pas indexée (pas de propagation).
		Écriture		= 3 transferts de page (en écriture) ≈ 6 ms
Projection P_2 sur $\{l.Titre, l.NomEdit\}$	Prod.	Parcours séquentiel	Oui	= 15 transferts de pages (en lecture) ≈ 15 ms
		Parcours indexé	Non	N/A : la relation d'entrée de la projection (Tmp_2) n'est pas indexée (pas de propagation).
		Écriture		= 9 transferts de page (en écriture) ≈ 18 ms
Jointure J	Prod.	Boucles imbriquées	Oui	= $3 + (3 \times 9)$ = 30 transferts de pages (en lecture) ≈ 30 ms
		Boucles optimisées	Oui	= $3 + ((3/43)^\wedge \times 9)$ = 12 transferts de pages (en lecture) ≈ 12 ms
		Boucle indexée	Non	N/A : la relation interne de la jointure (Tmp_4) n'est pas indexée (pas de propagation).
		Écriture		= 2 transferts de pages (en écriture) ≈ 4 ms
Projection P_3 sur $l.Titre$	Prod.	Parcours séquentiel	Oui	= 2 transferts de pages (en lecture) ≈ 2 ms
		Parcours indexé	Non	N/A : la relation d'entrée de la projection (Tmp_5) n'est pas indexée (pas de propagation).
		Écriture		= 1 transfert de page (en écriture) ≈ 2 ms

Complètement étiqueté, l'arbre de requête A_{51} est le suivant :



Cet étiquetage nous indique les points suivants :

- En théorie, on aurait pu associer à cet arbre de requête 96 plans d'exécution (2 pour chaque sélection et pour chaque projection et 3 pour la jointure),
- On peut effectivement lui associer 8 plans d'exécution,
- Le moins cher de ces 8 plans d'exécution coûte (121 TP_L + 45 TP_E) soit 211 ms ($\approx 0,2$ s).



Remarque

Ce coût est légèrement plus élevé que celui du meilleur des plans d'exécution associés à l'arbre A_{41} (dont le coût était de 187 ms). Ceci s'explique ici par le fait que les projections supplémentaires (faites entre les sélections et la jointure) ont un coût plus élevé que les gains qu'elles apportent en réduisant les relations à l'entrée de la jointure...

Cela ne sera cependant pas toujours le cas : il est donc tout de même globalement conseillé d'étudier les coûts liés aux descentes des projections.

9.2 Prise en compte de l'optimisation pendant l'évaluation : les pipelines

Tout ou partie de la mise en œuvre de l'évaluation d'un arbre de requête peut être réalisée en **pipeline**.



Définition : « pipeline »

Un **pipeline** est assimilable à un « canal » en mémoire de travail permettant de relier directement les zones de la mémoire de travail sur lesquelles opèrent 2 opérateurs relationnels successifs d'un arbre de requête. Ainsi, l'opérateur relationnel situé « à l'entrée » du pipeline transfert **directement** la relation temporaire qu'il produit à l'opérateur relationnel situé « à la sortie » du pipeline.

9.2.1 Principe

Globalement, au niveau d'une relation temporaire R_{tmp} produite par un opérateur relationnel $OpRel_1$ puis lue par un opérateur relationnel $OpRel_2$, on peut observer :

- Si R_{tmp} est transmise sans pipeline de $OpRel_1$ à $OpRel_2$:
 - L'opérateur $OpRel_1$ écrit son résultat R_{tmp} en mémoire cache puis le cache est vidé en mémoire de stockage avant d'évaluer l'opérateur $OpRel_2$: la relation R_{tmp} est donc écrite sur disque (ce qui provoque des transferts en écriture),
 - L'opérateur $OpRel_2$ lit la relation d'entrée R_{tmp} depuis la mémoire cache mais, celle-ci étant vide, la relation est transférée depuis la mémoire de stockage (en lecture, donc)

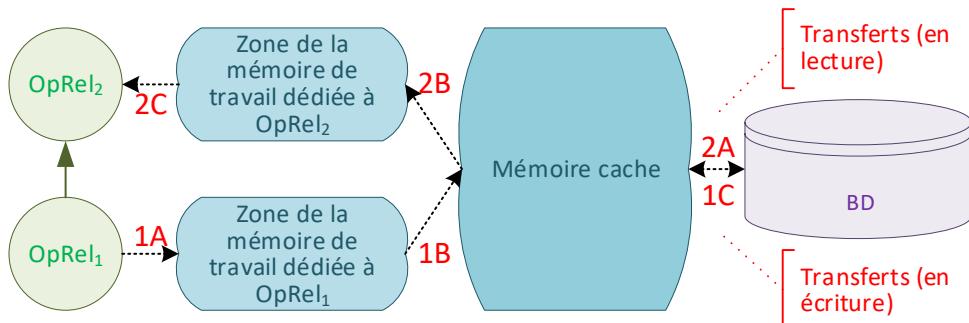


Figure 165. Transmission d'une relation temporaire sans pipeline

- Si R_{tmp} est transmise via un pipeline de $OpRel_1$ à $OpRel_2$:
 - L'opérateur $OpRel_1$ envoie son résultat R_{tmp} directement dans le pipeline : cette relation n'est donc pas écrite en mémoire cache et donc pas vidée en mémoire de stockage, le coût d'écriture de l'opérateur $OpRel_1$ lié à l'écriture de la relation R_{tmp} est donc annulé !
 - L'opérateur $OpRel_2$ lit la relation d'entrée R_{tmp} directement depuis le pipeline : cette relation n'est donc pas à charger (en lecture) depuis la mémoire de stockage, le coût de production de l'opérateur $OpRel_2$ lié à la lecture de la relation R_{tmp} est donc lui aussi annulé !

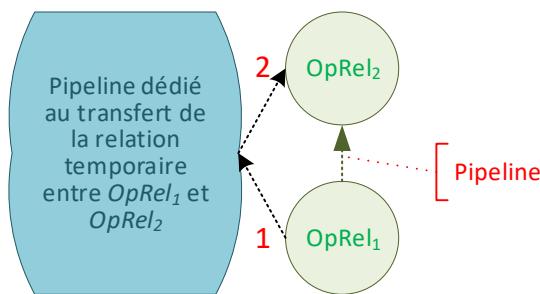


Figure 166. Transmission d'une relation temporaire avec pipeline

9.2.2 Mise en œuvre et prise en compte par le moteur d'optimisation

Cette technique de mise en place de pipelines pour évaluer un arbre de requête semble pouvoir être « la panacée » ! En effet, il suffit de se dire que la mise en œuvre d'un pipeline pour transférer chaque relation temporaire dans un arbre de requête suffit à annuler tous les coûts de production liés à la lecture de ces relations temporaires et tous les coûts d'écriture liés à l'écriture de ces mêmes relations temporaires : il ne resterait donc que le coût de production lié à la lecture des relations réelles et le coût d'écriture lié à l'écriture de la relation résultat !!! 😊 Mais non : cette technique n'est malheureusement pas la panacée espérée... 😞

Question

Pourquoi ? Cela semble vraiment idéal !



Réponse

En théorie, oui ! En pratique, cependant, la mise en place d'évaluations en pipelines complexifie énormément (de façon exponentielle) l'écriture du moteur d'évaluation de requêtes du SGBD. Cette complexification « à outrance » réduit vite les ardeurs des éditeurs de SGBD (même les plus gros) quant au nombre de pipelines supportés pendant l'évaluation d'un arbre de requête (voire même, pour certains, quant au support même du principe de l'évaluation en pipeline).

Ainsi, un moteur d'évaluation de requêtes peut, selon le SGBD, supporter 0, 1, 2, ..., n évaluations en pipeline en parallèle pour un arbre de requête.

Question

Que sont des « évaluations en pipeline faites en parallèles » ?



Réponse

Nous allons répondre en apportant une hypothèse simplificatrice basée sur un découpage d'un arbre de requête en niveaux :

- Pour chaque niveau niv , on regarde le nombre de relations temporaires $NbreIT_{mp}(niv)$ figurant à ce niveau,
- Parmi ces relations temporaires du niveau niv , on peut en évaluer au maximum $Nb_{pipelines}$ (les autres, s'il y en a, sont évaluées de façon classique, *i.e.* via la mémoire cache et la mémoire de stockage).



Remarques

Forcément, si $Nb_{pipelines} < NbreIT_{mp}(niv)$, cela démultiplie les possibilités à étudier, rien que pour le niveau niv ! Quand il se pose, le choix des branches à évaluer en pipeline est donc déterminant dans le choix du plan d'exécution à mettre en œuvre pour résoudre une requête.

En revanche, si $Nb_{pipelines} \geq NbreIT_{mp}(niv)$, toutes les relations temporaires du niveau niv seront évaluées en pipeline en parallèle.



Attention

La visibilité des niveaux dans un arbre de requête n'est bonne que si cet arbre a été correctement dessiné (donc pas « n'importe comment »).



Exemple

Reprendons l'arbre de requête A_{51} afin de le voir « en niveaux » :

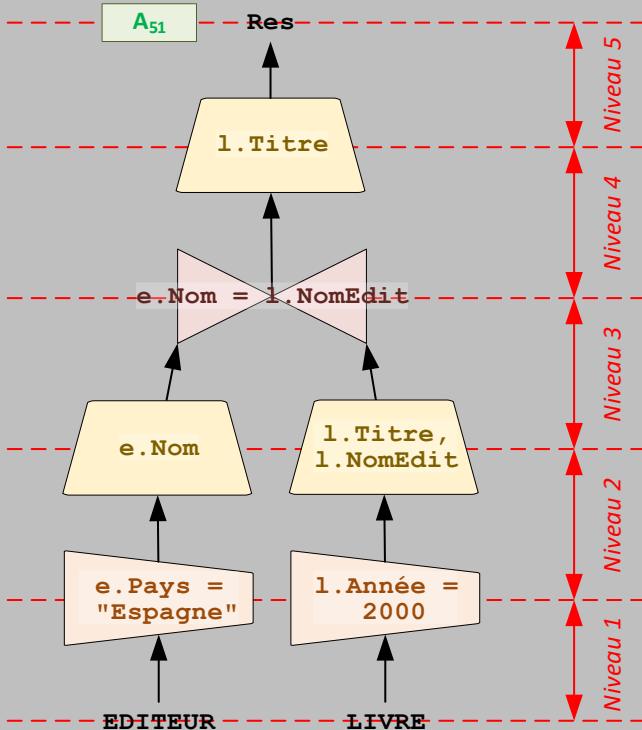


Figure 167. Niveaux dans l'arbre de requête A_{51} pour les évaluations en pipeline

Si le SGBD supporte « 1 pipeline simultané » :

- *Au niveau 1* : il n'y a aucune relation temporelle, donc rien à évaluer en pipeline,
- *Au niveau 2* : il faudra étudier le cas où le pipeline est mis en œuvre pour ce qui a trait aux éditeurs (branche de gauche) MAIS AUSSI le cas où le pipeline est mis en œuvre pour ce qui a trait aux livres (branche de droite),
- *Au niveau 3* : là encore, il faudra étudier le cas où le pipeline est mis en œuvre pour ce qui a trait aux éditeurs (branche de gauche) MAIS AUSSI le cas où le pipeline est mis en œuvre pour ce qui a trait aux livres (branche de droite),
- *Au niveau 4* : il n'y a qu'une seule relation temporelle, elle est donc évaluée en pipeline,
- *Au niveau 5* : il n'y a aucune relation temporelle, donc rien à évaluer en pipeline.

Si le SGBD supporte « 2 pipelines simultanés », cela ne change rien aux niveaux 1, 4 et 5. En revanche, aux niveaux 2 et 3, les 2 relations temporaires peuvent être simultanément transmises par ces pipelines. Enfin, si le SGBD supporte « 3 pipelines simultanés ou plus » : cela n'apporte rien de plus (mais le moteur d'évaluation a été énormément complexifié !).

Le moteur d'optimisation pré-évaluation connaît les capacités d'évaluation en pipeline offertes par le moteur d'évaluation : il en tient donc compte lors de l'analyse des plans d'exécution associés aux arbres de requête équivalents qu'il a pu construire.

9.2.3 Étude de cas (partielle)

Reprenez les arbres de requête A_{11} à A_{41} et étudions leur évaluation en pipeline (avec 1 puis 2 pipelines simultanés)...

9.2.3.1 Mise en œuvre d'une évaluation avec 1 pipeline

Voyons ce que cela donne si le SGBD est capable de mettre en œuvre 1 seul pipeline par niveau dans l'arbre de requête :

- Pour l'arbre de requête A_{11} : les relations temporaires Tmp_1 et Tmp_2 peuvent être passées en pipeline. La transmission en pipeline de la relation temporaire Tmp_1 annule totalement le coût d'écriture de la jointure et annule totalement le coût de production de la sélection (puisque la relation temporaire Tmp_1 est sa seule entrée). De même, la transmission en pipeline de la relation temporaire Tmp_2 annule totalement le coût d'écriture de la sélection et annule totalement le coût de production de la projection (puisque la relation temporaire Tmp_2 est sa seule entrée). Ainsi, le coût du meilleur des plans d'exécution associés effectivement à cet arbre de requête A_{11} évalué avec 1 pipeline est de $(3\ 750 \text{ TP}_L + 1 \text{ TP}_E) = 3\ 752 \text{ ms}$.

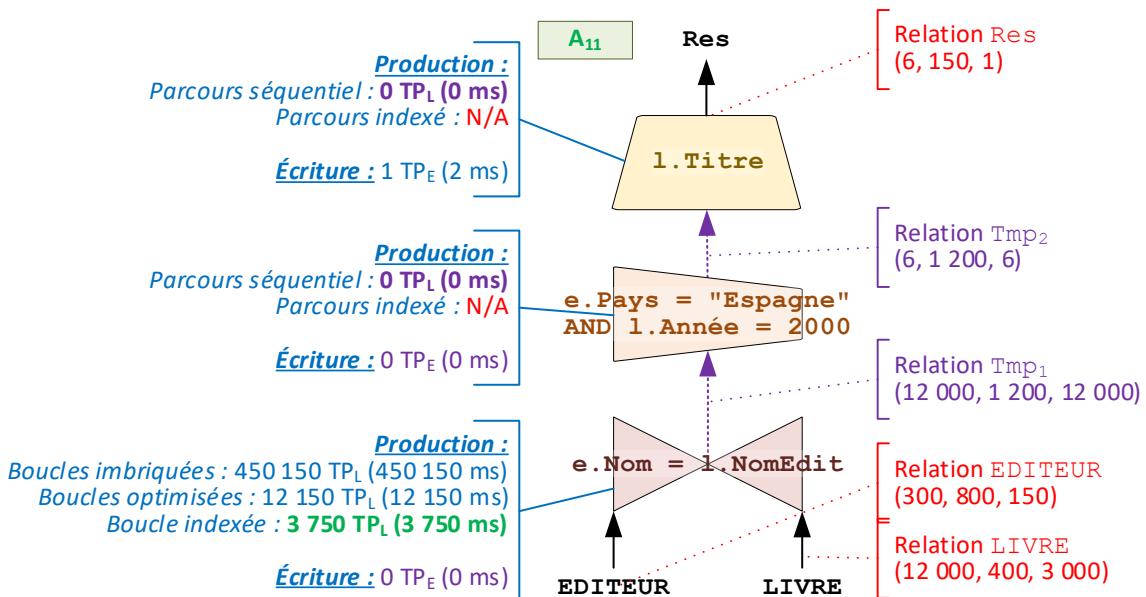


Figure 168. Optimisation (étude de cas) : prise en compte d'un pipeline pour l'arbre A_{11}

- Pour l'arbre de requête A_{21} : les relations temporaires Tmp_1 , Tmp_2 et Tmp_3 peuvent être passées en pipeline. La transmission en pipeline de la relation temporaire Tmp_1 annule totalement le coût d'écriture de la première sélection (sur $e.\text{Pays}$) et annule partiellement le coût de production de la jointure (la partie du coût de production qui est annulée est celle qui est due à la lecture de la relation externe). De même, la transmission en pipeline de la relation temporaire Tmp_2 annule totalement le coût d'écriture de la jointure et annule totalement le coût de production de la seconde sélection (sur $l.\text{Année}$, puisque la relation temporaire Tmp_2 est sa seule entrée). Enfin, la transmission en pipeline de la relation temporaire Tmp_3 annule totalement le coût d'écriture de cette seconde sélection et annule totalement le coût de production de la projection (puisque la relation temporaire Tmp_3 est sa seule entrée). Ainsi, le coût du meilleur des plans d'exécution associés effectivement à cet arbre de requête A_{21} avec 1 pipeline est de $(376 \text{ TP}_L + 1 \text{ TP}_E) = 378 \text{ ms}$.

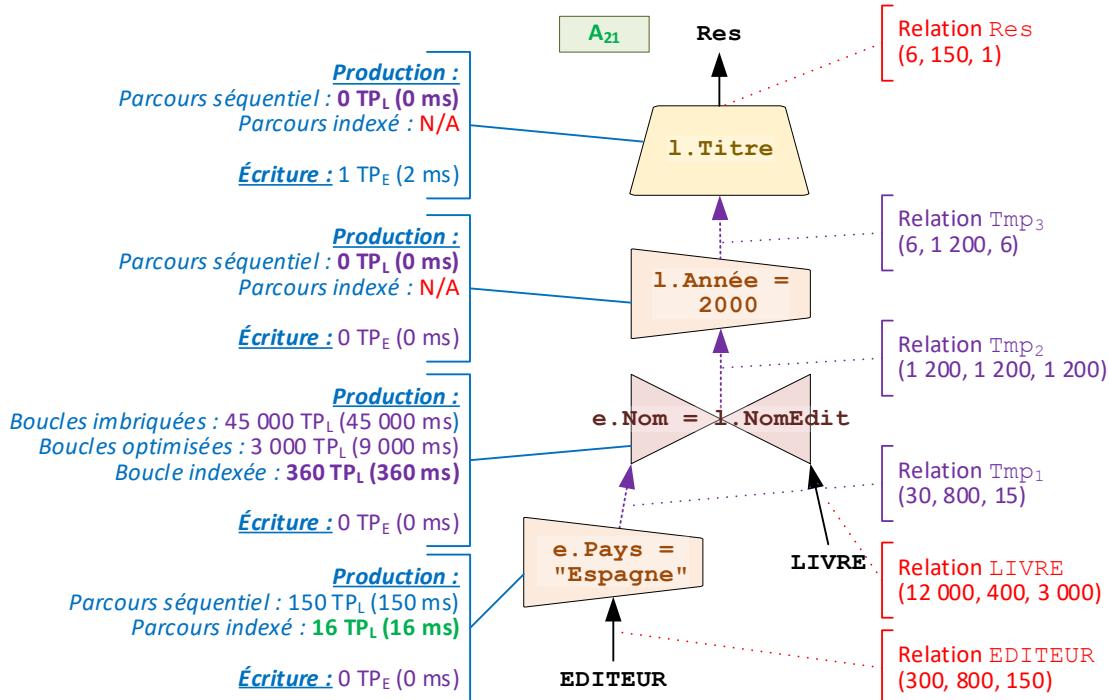


Figure 169. Optimisation (étude de cas) : prise en compte d'un pipeline pour l'arbre A₂₁

- Pour l'arbre de requête A₃₁ : la situation est analogue à celle de l'arbre de requête précédent sauf pour l'annulation partielle du coût de production de la jointure : cette fois, c'est la partie du coût de production due à la lecture de la relation externe qui est annulée. Ainsi, le coût du meilleur des plans d'exécution associés effectivement à cet arbre de requête A₃₁ avec 1 pipeline est de (211 TP_L + 1 TP_E) = 213 ms.

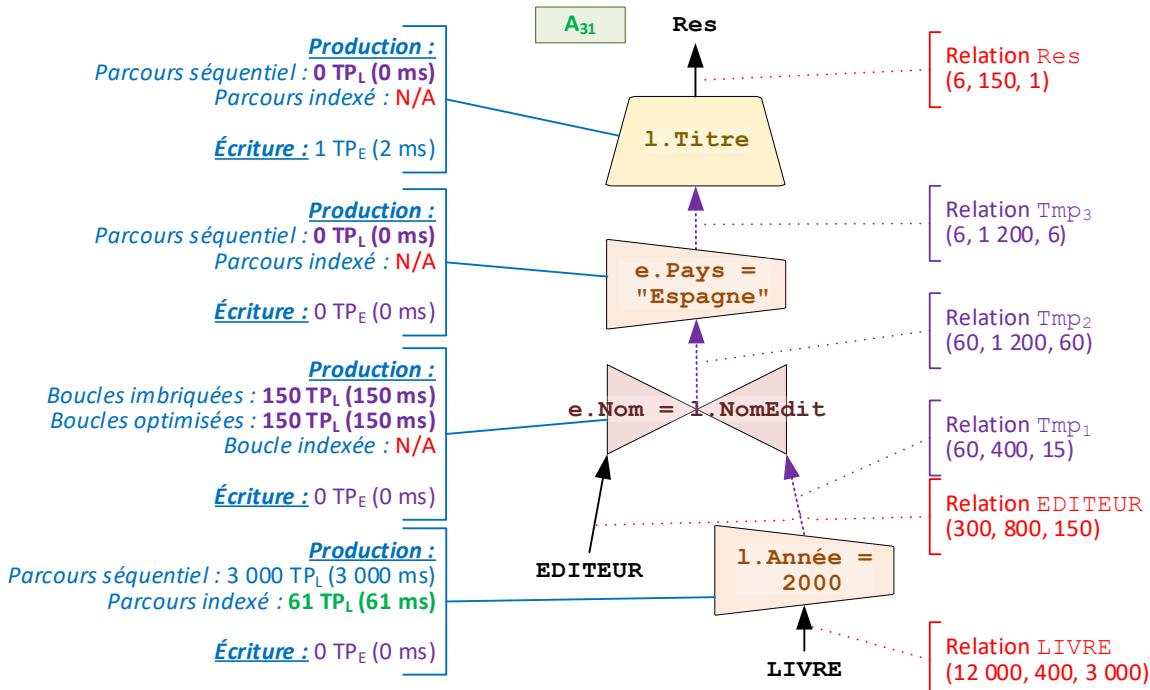


Figure 170. Optimisation (étude de cas) : prise en compte d'un pipeline pour l'arbre A₃₁

- Pour l'arbre de requête A₄₁ : cette fois on a 2 relations temporaires au même niveau (à l'entrée de la jointure), il faut donc étudier les 2 cas possibles...
 - Dans le cas où c'est la branche externe de la jointure qui est transmise en pipeline, le coût du meilleur des plans d'exécution associés effectivement à cet arbre de requête A_{41-A} avec 1 pipeline est de (92 TP_L + 16 TP_E) = 124 ms.

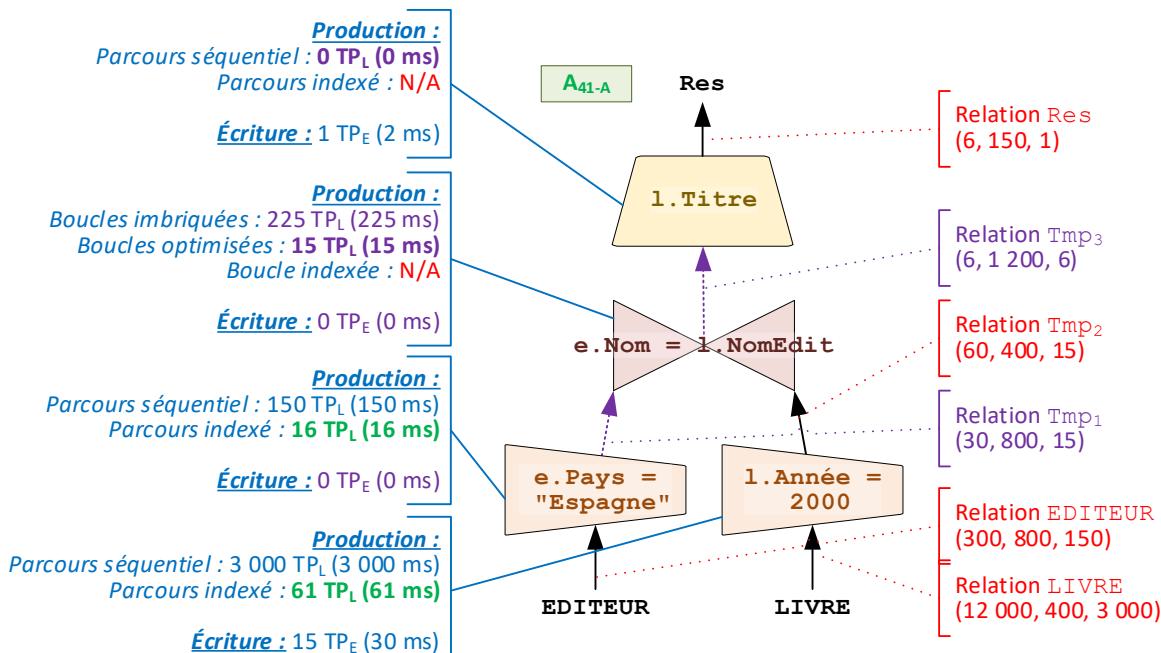


Figure 171. Optimisation (étude de cas) : prise en compte d'un pipeline pour l'arbre A₄₁ (cas 1)

- Dans le cas où c'est la branche interne de la jointure qui est transmise en pipeline, le coût du meilleur des plans d'exécution associés effectivement à cet arbre de requête A_{41-B} avec 1 pipeline est de (92 TP_L + 16 TP_E) = 124 ms.

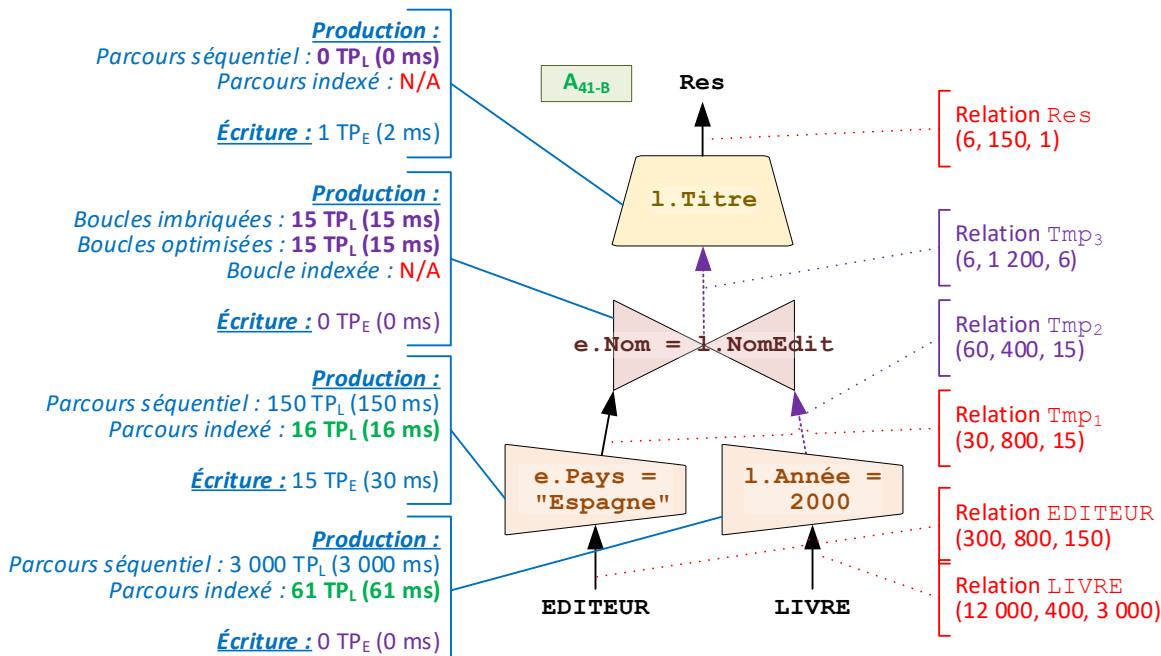


Figure 172. Optimisation (étude de cas) : prise en compte d'un pipeline pour l'arbre A₄₁ (cas 2)

9.2.3.2 Mise en œuvre d'une évaluation avec 2 pipelines

Voyons ce que cela donne si le SGBD est capable de mettre en œuvre 2 pipelines par niveau dans l'arbre de requête :

- Pour l'arbre de requête A_{11} : aucune nouvelle amélioration ne peut être apportée puisqu'il y a au plus 1 relation temporaire à chaque niveau de cet arbre de requête.
- Pour l'arbre de requête A_{21} : là non plus, on ne peut constater aucune amélioration (pour la même raison).
- Pour l'arbre de requête A_{31} : mêmes causes, mêmes effets.
- Pour l'arbre de requête A_{41} : ici, on peut transmettre simultanément à la jointure ses 2 entrées (et, donc, son coût de production est donc maintenant totalement annulé). Cette fois, le coût du meilleur des plans d'exécution associés effectivement à cet arbre de requête A_{41-C} avec 2 pipelines est de $(77 \text{ TP}_L + 1 \text{ TP}_E) = 79 \text{ ms}$.

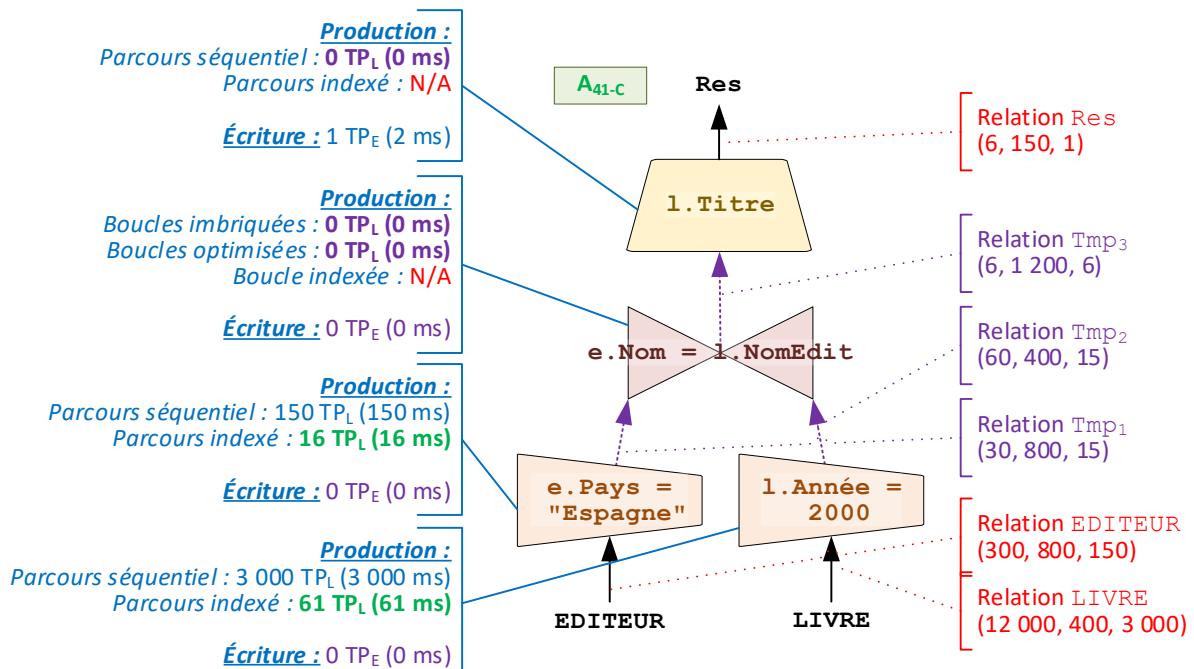


Figure 173. Optimisation (étude de cas) : prise en compte de 2 pipelines pour l'arbre A_{41}

9.2.3.3 Mise en œuvre avec 3 pipelines (ou plus)

Disposer de plus de 2 pipelines ne servirait ici à rien (en tous les cas pour ces arbres-là) : aucun d'entre eux ne fait en effet apparaître plus de 2 relations temporaires à un même niveau.

9.2.3.4 Bilan

On peut donc observer le bilan suivant :

Arbre de requête	Sans pipeline	Avec 1 pipeline		Avec 2 pipelines		
	Coût	Coût	Gain (/sans pipeline)	Coût	Gain (/sans pipeline)	Gain (/1 pipeline)
A₁₁	39 770 ms	3 752 ms	≈ 90,57%	3 752 ms	≈ 90,57%	0%
A₂₁	4 041 ms	378 ms	≈ 90,65%	378 ms	≈ 90,65%	0%
A₃₁	497 ms	213 ms	≈ 42,85%	213 ms	≈ 42,85%	0%
A₄₁	A_{41-A} A_{41-B}	187 ms	124 ms²¹² 124 ms²¹²	≈ 33,69% ≈ 33,69%	79 ms	≈ 57,75% ≈ 36,29%

Tableau 63. Bilan de la mise en œuvre d'1 ou 2 pipelines sur l'étude de cas

On observe déjà dans ce bilan que le passage à 2 pipelines soit ne provoque aucun gain (par rapport à la mise en place d'un seul), soit provoque un gain mais moindre (en valeur). Il est montré que la multiplication du nombre de pipelines procure en effet des gains de moins en moins élevés (voir nuls) alors que, comme cela a déjà été dit, cela complexifie énormément le moteur d'évaluation du SGBD... On comprend mieux que les éditeurs rechignent à les multiplier à l'envi !

9.3 Paramètres liés à l'optimisation de requêtes

Les paramètres liés à l'optimisation de requêtes sont les suivants :

Notation	Unité(s) usuelle(s)	Paramètre
-	-	Stratégie de propagation des index au-delà des opérateurs relationnels : pas de propagation, seulement quelques opérateurs relationnels (ils sont alors indiqués) ou tous les opérateurs relationnels.
-	-	Possibilité d'évaluation en pipeline : non ou oui (le nombre de pipelines simultanés au maximum est alors indiqué).
-	-	Introduction des relations dans les arbres de requête initiaux (quand cela est possible, étant donné qu'il faut tout de même respecter les jointures faites dans ladite requête !) : par ordre croissant de leur cardinalité, par ordre d'apparition dans la requête, ...
-	-	Heuristiques supportées (et indication de leur caractère optionnel/obligatoire) : <ul style="list-style-type: none"> • ALGJ : non, optionnellement (gauche et/ou droite), obligatoirement (gauche et/ou droite), • DSP : non, optionnellement, obligatoirement.
-	-	Contraintes éventuelles sur le type d'arbres de requêtes construits (initiaux ET/OU équivalents) : liste de ces contraintes particulières éventuelles...

Tableau 64. Paramètres liés à l'optimisation de requêtes

²¹² C'est vraiment un hasard que les coûts des arbres A_{41-A} et A_{41-B} (évalués avec 1 pipeline) soient identiques.

Stratégies de gestion de données relationnelles

Synthèse

10 Récapitulatif de l'ensemble des paramètres

SOMMAIRE DÉTAILLÉ DU CHAPITRE 10

10.1	Paramètres liés au contrôle des données (livre 1).....	331
10.1.1	Paramètres liés aux transactions.....	331
10.1.2	Paramètres liés à la journalisation	331
10.2	Paramètres liés à l'optimisation (livre 2)	331
10.2.1	Paramètres de la machine/de l'OS	331
10.2.1.1	Au niveau physique de la mémoire de stockage (secteurs).....	332
10.2.1.2	Au niveau logique des mémoires de stockage et de travail (blocs)	332
10.2.2	Au niveau logique des mémoires de stockage et de travail (pages)	333
10.2.3	Paramètres du SGBD	334
10.2.3.1	Impacts des hypothèses simplificatrices d'adressage des n-uplets	334
10.2.3.2	Paramètres liés au stockage des n-uplets des relations	335
10.2.3.2.1	Paramètres de stockage généraux au SGBD (indépendants de toute relation R)	335
10.2.3.2.2	Paramètres de stockage de n-uplets propres à une relation R	336
10.2.3.2.3	Paramètres de stockage de PS ou de la TC d'une relation R	337
10.2.3.2.4	Quantité « d'informations » par page de stockage d'une relation R	338
10.2.3.2.5	Nombre de pages de stockage « d'informations » pour une relation R	339
10.2.3.3	Paramètres liés à la mémoire cache (forcément généraux au SGBD).....	339
10.2.3.4	Paramètres liés aux index I(R, c).....	340
10.2.3.4.1	Paramètres liés aux n-uplets de la relation indexée R	340
10.2.3.4.2	Paramètres généraux (indépendants de la structure) de l'index I(R, c)	341
10.2.3.4.3	Paramètres spécifiques aux arbres B+	342
10.2.3.4.3.1	Paramètres liés au contenu d'un arbre B+	342
10.2.3.4.3.2	Paramètres liés aux performances d'un arbre B+.....	343
10.2.3.4.4	Paramètres spécifiques aux IMCAHSaR	343
10.2.3.4.4.1	Paramètres liés au contenu des IMCAHSaR.....	343
10.2.3.4.4.2	Paramètres liés aux performances d'un IMCAHSaR	344
10.2.3.4.5	Paramètres spécifiques aux IMCAHSsR.....	345
10.2.3.4.6	Paramètres spécifiques aux IMCAHD.....	346
10.2.3.5	Paramètres liés à l'optimisation	347

FIGURES DU CHAPITRE 10

Aucune entrée de table d'illustration n'a été trouvée.

TABLEAUX DU CHAPITRE 10

Aucune entrée de table d'illustration n'a été trouvée.

ÉQUATIONS DU CHAPITRE 10

Aucune entrée de table d'illustration n'a été trouvée..

Ce « chapitre » résume l'ensemble des paramètres (fournis ou calculés) potentiellement impliqués dans le cadre de ce cours. Ils sont regroupés en « familles »...

10.1 Paramètres liés au contrôle des données (livre 1)

10.1.1 Paramètres liés aux transactions

Notation	Unité(s) usuelle(s)	Paramètre
-	-	Mode de mise à jour de la base de données : immédiate ou différée .
-	-	Type de transactions : non-contraintes , à 2 phases ou « spéciales ».
-	-	Opérations de reclassement de verrous supportées ? Non , Partiellement (voir les « limitations imposées » ci-dessous), Totalement .
-	-	Stratégie de résolution des problèmes d'accès concurrentiels : simple ou évoluée .
-	-	Stratégie de résolution des problèmes d'interblocage : détection ou prévention .
-	-	Limitations supplémentaires imposées : des limitations particulières peuvent être imposées et sont alors décrites ici.

10.1.2 Paramètres liés à la journalisation

Aucun paramètre spécifique n'est requis (le mode de mise à jour dépend du gestionnaire transactionnel, cf. §10.1.1).

10.2 Paramètres liés à l'optimisation (livre 2)

10.2.1 Paramètres de la machine/de l'OS



Remarque

L'ensemble de ces paramètres étant implicitement indépendant du SGBD (*a fortiori* des relations manipulées), si certains sont à calculer, il n'est nécessaire de les calculer qu'une et une seule fois pour chaque mémoire de stockage *MS* impliquée ! D'autant plus qu'ils ont tous une valeur fixe (par mémoire de stockage *MS*) !



En pratique

Ces paramètres, sont rarement indiqués (c'est pourquoi ils sont écrits en gris dans le tableau ci-dessous) : en raison des hypothèses simplificatrices que nous adopterons, ils seront en effet rarement utiles (si c'est le cas, ce sera pour retrouver un autre paramètre qui ne nous aurait pas été directement fourni).

10.2.1.1 Au niveau physique de la mémoire de stockage (secteurs)

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$tps_{lectSecteur}^{fixe}(MS)$	ms	Temps de lecture (chargement) d'un secteur depuis la mémoire de stockage MS (considéré fixe pour une mémoire de stockage donnée).
$tps_{ecritSecteur}^{fixe}(MS)$	ms	Temps d'écriture (déchargement) d'un secteur vers la mémoire de stockage MS (considéré fixe pour une mémoire de stockage donnée).
$T_{secteur}^{fixe}(MS)$	octets	Taille d'un secteur de la mémoire de stockage MS (considérée fixe pour une mémoire de stockage donnée).
$nb_{secteurs}^{fixe}(MS)$	secteurs	Nombre (fixe) de secteurs utilisables sur la mémoire de stockage MS (au total, i.e. qu'ils soient « libres » ou déjà « occupés »).
$T_{idSecteur}^{fixe}(MS)$	octets	Taille d'un identificateur de secteur sur la mémoire de stockage MS (considérée fixe pour une mémoire de stockage donnée).

10.2.1.2 Au niveau logique des mémoires de stockage et de travail (blocs)

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$tps_{lectBloc}^{fixe}(Mem, OS)$	ms	Temps de lecture (chargement) d'un bloc (considéré fixe pour une mémoire, de stockage ou de travail, donnée et un OS donné).
$tps_{ecritBloc}^{fixe}(Mem, OS)$	ms	Temps d'écriture (déchargement) d'un bloc (considéré fixe pour une mémoire, de stockage ou de travail, donnée et un OS donné).
$T_{bloc}^{fixe}(Mem, OS)$	octets	Taille d'un bloc (considérée fixe pour une mémoire, de stockage ou de travail, donnée et un OS donné) ²¹³ : $T_{bloc}^{fixe}(MS, OS) = 2^{i(MS, OS)} \times T_{secteur}^{fixe}(MS)$ <p>(avec $i(MS, OS)$ un entier positif ou nul)</p>
$nb_{secteurs/bloc}^{fixe}(MS, OS)$	secteurs par bloc	Nombre (fixe pour une mémoire, forcément de stockage, et un OS donné) de secteurs occupés par un bloc : $nb_{secteurs/bloc}^{fixe}(MS, OS) = \frac{T_{bloc}^{fixe}(MS, OS)}{T_{secteur}^{fixe}(MS)}$
$nb_{blocs}^{fixe}(Mem, OS)$	blocs	Nombre (fixe) de blocs utilisables sur une mémoire, de stockage ou de travail, donnée via l'OS (au total, i.e. qu'ils soient « libres » ou déjà « occupés ») : $nb_{blocs}^{fixe}(MS, OS) = nb_{secteur}^{fixe}(MS) \times nb_{secteurs/bloc}^{fixe}(MS, OS)$
$T_{idBloc}^{fixe}(OS)$	octets	Taille d'un identificateur de bloc (considérée fixe pour un OS donné).

²¹³ La valeur de i est propre au couple mémoire de stockage/système d'exploitation.

10.2.2 Au niveau logique des mémoires de stockage et de travail (pages)

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$tps_{lectPage}^{fixe}(Mem, OS)$	ms	Temps de lecture d'une page depuis la mémoire, de travail ou de stockage (considéré fixe pour une mémoire et un OS donnés). Négligeable pour une mémoire de travail !
$tps_{ecritPage}^{fixe}(Mem, OS)$	ms	Temps d'écriture d'une page depuis la mémoire, de travail ou de stockage (considéré fixe pour une mémoire et un OS donnés). Négligeable pour une mémoire de travail !
$T_{page}^{fixe}(OS)$	octets	Taille d'une page (considérée fixe pour un OS donné) : $T_{page}^{fixe}(OS) = 2^{i(OS)} \times T_{secteur}^{fixe}(MS)$ $\text{et } T_{bloc}^{fixe}(MS, OS) = 2^{j(OS)} \times T_{page}^{fixe}(OS)$ $(avec T_{secteur}^{fixe}(MS) \leq T_{page}^{fixe}(OS) \leq T_{bloc}^{fixe}(MS, OS), i(OS) \text{ et } j(OS) \text{ des entiers positifs ou nuls})$
$nb_{pages}^{fixe}(Mem, OS)$	pages	Nombre (fixe) de pages utilisables sur la mémoire (stockage ou travail) par l'OS (« libres » ou « occupés »).
$nb_{secteurs/page}^{fixe}(MS, OS)$	secteurs par page	Nombre (fixe pour une mémoire <u>de stockage</u> et un OS donné) de secteurs « occupés » par une page : $nb_{secteurs/page}^{fixe}(MS, OS) = \frac{T_{page}^{fixe}(MS, OS)}{T_{secteur}^{fixe}(MS)}$
$nb_{pages/bloc}^{fixe}(OS)$	pages par bloc	Nombre (fixe pour un OS donné) de pages au sein d'un bloc : $nb_{pages/bloc}^{fixe}(MS, OS) = \frac{T_{bloc}^{fixe}(MS, OS)}{T_{page}^{fixe}(OS)}$
$nb_{pages}^{fixe}(Mem, OS)$	blocs	Nombre (fixe) de pages utilisables sur la mémoire (stockage ou travail) via l'OS (« libres » ou « occupées ») : $nb_{pages}^{fixe}(Mem, OS) = nb_{blocs}^{fixe}(Mem) \times nb_{pages/bloc}^{fixe}(OS)$
$T_{idPage}^{fixe}(OS)$	octets	Taille d'un identificateur de page (considérée fixe pour un OS donné).

10.2.3 Paramètres du SGBD



Remarque

Il est intéressant de noter que :

- Le calcul des paramètres indépendants d'une relation (**indiqués par une pseudo-relation notée « * »**) peut se faire une et une seule fois pour tout le SGBD : ils sont en effet valides pour tout le SGBD !
- De même, le calcul des paramètres liés à une relation R mais indiqués comme « fixes » peuvent être faits une et une seule fois pour la relation R : ils sont en effet valides pour cette relation tant que son modèle conceptuel n'est PAS modifié ET tant que les paramètres généraux de la machine/de l'OS/du SGBD, dont ils peuvent dépendre, ne sont pas modifiés non plus.

10.2.3.1 Impacts des hypothèses simplificatrices d'adressage des n-uplets

Notation	Unité(s) usuelle(s)	Paramètre <u>impacté</u>
$T_{page}^{fixe}(OS)$	octets	<p>La taille d'un bloc et d'une page sont les mêmes (notez que, du coup, la taille d'un bloc devient aussi implicitement indépendante d'une mémoire) :</p> $T_{page}^{fixe}(OS) = T_{bloc}^{fixe}(Mem, OS)$
$tps_{lectPage}^{fixe}(MS, OS)$	ms	<p>On fera des chargements et des déchargements de pages (et non plus de blocs stricto sensu) depuis/vers la mémoire de stockage :</p> $tps_{lectPage}^{fixe}(Mem, OS) = tps_{lectBloc}^{fixe}(Mem, OS)$ $tps_{ecritPage}^{fixe}(Mem, OS) = tps_{ecritBloc}^{fixe}(Mem, OS)$
$tps_{ecritPage}^{fixe}(MS, OS)$	ms	Cela ne change rien aux temps de chargements/déchargements de pages au niveau de la mémoire de travail : ils sont toujours considérés comme négligeables (<i>i.e.</i> nuls)...

10.2.3.2 Paramètres liés au stockage des n-uplets des relations

10.2.3.2.1 Paramètres de stockage généraux au SGBD (indépendants de toute relation R)

Notation	Unité(s) usuelle(s)	Paramètre (fourni)
-	-	Stratégie d'adressage des n-uplets (direct ou indirect).
-	-	Stratégie de stockage des valeurs d'attributs (fixe ou variable).
-	-	Stratégie de gestion du répertoire des déplacements situé à la fin des pages de stockage des n-uplets (statique ou dynamique).
$T_{idCaseDepl}^{fixe}(*)$	octets	Taille (fixe) d'un indice d'une case d'un répertoire des déplacements situé à la fin des pages de stockage des n-uplets.
$T_{caseDepl}^{fixe}(*)$	octets	Taille (fixe) d'une case d'un répertoire des déplacements situé à la fin des pages de stockage des n-uplets.
$nb_{casesDepl}^{fixe}(*)$	cases du rép. des dépl.	En gestion statique du répertoire des déplacements uniquement : nombre (fixe) de cases contenues dans le répertoire des déplacements situé à la fin des pages de stockage des n-uplets.
$T_{tailleValAttr}^{fixe}(*)$	octets	En format variable de stockage des valeurs d'attributs uniquement : place (fixe) occupée par l'indication de la taille effective d'une valeur d'attribut.
$T_{caractere}^{fixe}(*)$	octets	Place (fixe) occupée par un caractère (utile si on nous donne la taille des valeurs d'attributs en nombre de caractères et non en octets).
$T_{drapeauNUPS}^{fixe}(*)$	octets	En adressage direct uniquement : taille (fixe) d'un drapeau indiquant si ce qui suit correspond au stockage d'un n-uplet ou à celui d'un pointeur de suivi.
$T_{idLog}^{fixe}(*)$	octets	En mode d'adressage indirect uniquement : taille (fixe) d'un identificateur logique de n-uplet.

10.2.3.2.2 Paramètres de stockage de n-uplets propres à une relation R

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$Card(R)$	n-uplets	Cardinalité de (<i>i.e.</i> nombre de n-uplets contenus dans) la relation R : $Card(R) \equiv nb_{NU}^{exact}(R)$
$nb_{attributs}^{fixe}(R)$	attributs	Nombre d'attributs définis dans le modèle (intension) de la relation R .
$T_{totaleNU}^{fixe moy min max}(R)$	octets	Taille totale (fixe ou moyenne ou minimale ou maximale) d'un n-uplet de la relation R : $T_{totaleNU}^{fixe moy min max}(R) = T_{donneesNU}^{fixe moy min max}(R) + T_{metaNU}^{fixe}(R)$
$T_{donneesNU}^{fixe moy min max}(R)$	octets	Taille (fixe ou moyenne ou minimale ou maximale) des données « brutes » d'un n-uplet de la relation R : $T_{donneesNU}^{fixe moy min max}(R) = \sum_{i=1}^{i \leq nb_{attributs}^{fixe}(R)} T_{valAttr}^{fixe moy min max}(R, att_i)$
$T_{valAttr}^{fixe moy min max}(R, att)$	octets ou caractères	Taille (fixe ou moyenne ou minimale ou maximale) des valeurs de l'attribut att dans les n-uplets de la relation R . <div style="border: 1px solid #ccc; padding: 5px; background-color: #f0f0f0;"> Attention Si elle est donnée en caractères, n'oubliez pas de la « convertir » afin de l'obtenir en octets !!! Pour cela, multipliez la taille donnée en caractères par le nombre d'octets occupé par chaque caractère (cf. Tableau 28) qui vous sera alors forcément indiqué. </div>
$T_{metaNU}^{fixe}(R)$	octets	Taille (fixe) des métadonnées des n-uplets de la relation R . La nature de ces métadonnées (et donc leur taille) dépend des choix stratégiques faits ici : $T_{metaNU}^{fixe}(R) = \underbrace{T_{drapeauNUPS}^{fixe}(*)}_{si adressage direct} + \underbrace{\left(T_{tailleValAttr}^{fixe}(*) \times nb_{attributs}^{fixe}(R) \right)}_{si format variable} + \underbrace{T_{caseDepl}^{fixe}(*)}_{si gestion dynamique}$

10.2.3.2.3 Paramètres de stockage de PS ou de la TC d'une relation R

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$T_{totalePS}^{fixe}(R)$	octets	En mode d'adressage direct uniquement : taille totale (fixe) des pointeurs de suivi : $T_{totalePS}^{fixe}(R) = T_{donneesPS}^{fixe}(*) + T_{metaPS}^{fixe}(R)$
$T_{donneesPS}^{fixe}(*)$	octets	En mode d'adressage direct uniquement : taille (fixe) des données « brutes » d'un pointeur de suivi : $T_{donneesPS}^{fixe}(*) = T_{idPage}^{fixe}(*) + T_{idCaseDepl}^{fixe}(*)$
$T_{metaPS}^{fixe}(R)$	octets	En mode d'adressage direct uniquement : taille (fixe) des métadonnées d'un pointeur de suivi : $T_{metaPS}^{fixe}(R) = T_{drapeauNUPS}^{fixe}(*) + \underbrace{T_{caseDepl}^{fixe}(*)}_{\text{si gestion dynamique}}$
$nb_{PS}^{fixe moy min max}(R)$	pointeurs de suivi	En mode d'adressage direct uniquement : nombre total (fixe ou moyen ou minimal ou maximal) de pointeurs de suivi stockés avec les n-uplets de la relation R. On a forcément le bornage suivant : $0 \leq nb_{PS}^{fixe moy min max}(R) \leq Card(R)$
$nb_{PS/NU}^{fixe moy min max}(R)$	pointeurs de suivi par n-uplet	En mode d'adressage direct uniquement : nombre (fixe ou moyen ou minimal ou maximal) de pointeurs de suivi par n-uplet de la relation R ²¹⁴ : $nb_{PS/NU}^{fixe moy min max}(R) \xrightarrow{\text{ou}} \frac{Card(R)}{nb_{PS}^{fixe moy min max}(R)}$
$T_{ligneTC}^{fixe}(*)$	octets	En mode d'adressage indirect uniquement : taille (fixe) d'une ligne de la table de correspondance de la relation R : $T_{ligneTC}^{fixe}(*) = (T_{idPage}^{fixe}(*) + T_{idCaseDepl}^{fixe}(*)) + T_{idLog}^{fixe}(*)$
$nb_{lignesTC}^{exact}(R)$	lignes de la TC	En mode d'adressage indirect uniquement : nombre de lignes contenues dans la table de correspondance de la relation R : $nb_{lignesTC}^{exact}(R) = Card(R)$

²¹⁴ Il faudra « faire des tests » avec l'arrondi par défaut ET avec l'arrondi par excès.

10.2.3.2.4 Quantité « d'informations » par page de stockage d'une relation R

On désigne ici comme « information » les n-uplets (dans tous les cas) et les pointeurs de suivi (si on est en mode d'adressage direct) ou la table de correspondance (si on est en mode d'adressage indirect).

Notation	Unité(s) usuelle(s)	Paramètre (calculé)
$T_{utileDsPageNU}^{fixe}(*)$	octets	Taille (fixe) « utile » dans une page de stockage des déplacements (i.e. place utilisable dans la page pour y stocker des n-uplets et d'éventuels pointeurs de suivi) : $T_{utileDsPage}^{fixe}(*) = T_{page}^{fixe}(*) - \underbrace{(nb_{casesDepl}^{fixe}(*) \times T_{caseDepl}^{fixe}(*))}_{\text{si gestion statique uniquement}}$
$nbMax_{NU/page}^{moy max}(R)$	n-uplets par page	Nombre (moyen ou maximal) de n-uplets au plus par page : $\frac{T_{utileDsPageNU}^{fixe}(*)}{T_{NU}^{fixe moy min max}(R) + \underbrace{(nb_{PS/NU}^{fixe moy min max}(R) \times T_{PS}^{fixe}(R))}_{=0 \text{ si adressage indirect}}}$
$nbMax_{PS/page}^{moy max}(R)$	pointeurs de suivi par page	En mode d'adressage direct uniquement : nombre (moyen ou maximal) de pointeurs de suivi au plus par page ²¹⁵ : $nbMax_{PS/page}^{moy max}(R) \stackrel{\text{ou}}{=} nb_{NU/page}^{moy max}(R) \times nb_{PS/NU}^{fixe moy min max}(R)$ $=0 \text{ si adressage indirect}$
$nbMax_{lignesTC/page}^{fixe}(*)$	lignes de la TC par page	En mode d'adressage indirect uniquement : nombre maximal (forcément) de lignes de la TC au plus par page : $nbMax_{lignesTC/page}^{fixe}(*) \stackrel{\text{ou}}{=} \frac{T_{page}^{fixe}(*)}{T_{ligneTC}^{fixe}(*)}$

²¹⁵ Il faudra « faire des tests » avec l'arrondi par défaut ET avec l'arrondi par excès.

10.2.3.2.5 Nombre de pages de stockage « d'informations » pour une relation R

Notation	Unité(s) usuelle(s)	Paramètre (calculé)
$nb_{pagesNU}^{moy min}(R)$	pages	<p>Nombre (moyen ou minimal) de pages de stockage de n-uplets nécessaire pour stocker tous les n-uplets (ET les éventuels pointeurs de suivi en mode d'adressage direct) de la relation R :</p> $nb_{pagesNU}^{moy max}(R) \stackrel{\text{Card}(R)}{=} \frac{nbMax_{NU/page}^{moy max}(R)}{}$ <div style="text-align: center;">  </div> <p>Remarque Les éventuels pointeurs de suivi ont déjà été « comptés » : ils ont été pris en compte dans le calcul de nombre de n-uplets par page, il ne faut donc PAS les compter ici une 2^{nde} fois !</p>
$nb_{pagesTC}^{exact}(R)$	pages	<p>En mode d'adressage indirect uniquement : nombre (forcément exact) de pages nécessaires pour stocker toutes les lignes de la table de correspondance associée à la relation R :</p> $nb_{pagesTC}^{exact}(R) \stackrel{\text{Card}(R)}{=} \frac{nb_{lignesTC}^{exact}(R)}{nbMax_{lignesTC/page}^{fixe} (*)}$
$nb_{pages}^{moy min}(R)$	pages	<p>Nombre (moyen ou minimal) de pages occupées au total pour le stockage des « informations » (i.e. n-uplets et (pointeurs de suivi OU table de correspondance)) de la relation R :</p> $nb_{pages}^{moy min}(R) = nb_{pagesNU}^{moy min}(R) + nb_{pagesTC}^{exact}(R)$ <p style="color: red; text-align: center;"><i>si adressage indirect uniquement</i></p>

10.2.3.3 Paramètres liés à la mémoire cache (forcément généraux au SGBD)

Notation	Unité(s) usuelle(s)	Paramètre (fourni)
-	-	Stratégie de rejet de pages (LRU ou MRU).
-	-	Stratégie de gestion des conflits de rejets de pages (prédictive ou hasardeuse).
-	-	Possibilité de punaisage de pages ? (oui ou non)
$T_{cache}^{fixe} (*)$	pages ou cases du cache	Taille (fixe) de la mémoire cache (nombre de pages/cases qu'elle peut contenir au maximum).
$T_{utilleDsCache}^{exacte} (*)$	pages ou cases du cache	<p>Nombre (exact) de pages/cases « libres » dans le cache (i.e. non-occupées par des pages punaisées) :</p> $T_{utilleDsCache}^{exacte} (*) = T_{cache}^{fixe} (*) - Nb_{pagesPunaisees}^{exact}(*)$

10.2.3.4 Paramètres liés aux index $I(R, c)$

10.2.3.4.1 Paramètres liés aux n-uplets de la relation indexée R

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$Cout_{chargementNU}^{exact moy min}(R, c = v)$	pages	<p>Nombre (exact ou moyen ou minimal) de pages à charger pour lire tous les n-uplets d'une relation R tels que leur constituant c vaille une valeur v une fois les adresses logiques de ces n-uplets récupérées dans l'index $I(R, c)$:</p> <ul style="list-style-type: none"> • Si l'index $I(R, c)$ est groupé : $Cout_{chargementNU}^{moy min}(R, c = v) \geq \frac{nb_{pages}^{moy min}(R)}{nb_{valDiff}^{exact}(R, c)}$ <ul style="list-style-type: none"> • Si l'index $I(R, c)$ est non-groupé : $Cout_{chargementNU}^{exact}(R, c = v) \geq \frac{Card(R)}{nb_{valDiff}^{exact}(R, c)}$
$T_{adrLogNU}^{fixe}(*)$	octets	<p>Taille (forcément fixe) d'une adresse logique de n-uplet :</p> <ul style="list-style-type: none"> • Si on est en mode d'adressage direct pour le stockage des n-uplets : $T_{adrLogNU}^{fixe}(*) = T_{idPage}^{fixe}(*) + T_{idCaseDepl}^{fixe}(*)$ <ul style="list-style-type: none"> • Si on est en mode d'adressage indirect pour le stockage des n-uplets : $T_{adLogNU}^{fixe}(*) = T_{idLog}^{fixe}(*)$

10.2.3.4.2 Paramètres généraux (indépendants de la structure) de l'index $I(R, c)$

Notation	Unité(s) usuelle(s)	Paramètre (fourni)
$nb_{adrLogNU/entree}^{fixe moy}(I(R, c))$	adresses logiques de n-uplets par entrée d'index	<p>Nombre (fixe ou moyen) d'adresses logiques de n-uplets par entrée d'index :</p> <ul style="list-style-type: none"> • Si $I(R, c)$ est primaire : $nb_{adrLogNU/entree}^{fixe}(I(R, c)) = 1$ <ul style="list-style-type: none"> • Si $I(R, c)$ est secondaire : $nb_{adrLogNU/entree}^{moy}(I(R, c)) \stackrel{\text{def}}{=} \frac{Card(R)}{nb_{valDiff}^{exact}(R, c)}$
$T_{entree}^{fixe moy min max}(I(R, c))$	octets	<p>Taille (fixe ou moyenne ou minimale ou maximale) d'une entrée d'index dans l'index $I(R, c)$:</p> $T_{entree}^{fixe moy min max}(I(R, c)) = T_{valAtt}^{fixe moy min max}(R, att)$ $+ \left(nb_{adrLogNU/entree}^{fixe moy}(I, (R, c)) \times \underbrace{T_{adrLogNU}^{fixe}(*)}_{\text{dépend du mode d'adressage}} \right)$
$nb_{entrees}^{exact}(I(R, c))$	entrées d'index	<p>Nombre (exact) d'entrées dans l'index $I(R, c)$:</p> <ul style="list-style-type: none"> • Si l'index est primaire : $nb_{entrees}^{exact}(I(R, c)) = Card(R)$ <ul style="list-style-type: none"> • Si l'index est secondaire : $nb_{entrees}^{exact}(I(R, c)) = nb_{valDiff}^{exact}(R, c)$
$nbMax_{entrees/page}^{fixe moy min max}(I(R, c))$	entrées d'index par page	<p>Nombre (fixe ou moyen ou minimal ou maximal) d'entrées d'index au plus par page : ce paramètres dépend de la structure de l'index $I(R, c)$.</p>
$nb_{pagesEntrees}^{fixe moy min max}(I(R, c))$	pages	<p>Nombre (fixe ou moyen ou minimal ou maximal) de pages nécessaires au stockage des entrées d'index (on ne compte donc PAS ici les pages stockant les informations de circulation dans l'index) :</p> $nb_{pagesEntrees}^{fixe moy min max}(I(R, c)) \stackrel{\text{def}}{=} \frac{nb_{entrees}^{exact}(I(R, c))}{nbMax_{entrees/page}^{fixe moy min max}(I(R, c))}$

10.2.3.4.3 Paramètres spécifiques aux arbres B+

10.2.3.4.3.1 Paramètres liés au contenu d'un arbre B+

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$e_{max}(I(R, c))$	-	Ordre de l'arbre B+.
$nbMax_{entrees/page}^{fixe moy min max}(I(R, c))$	entrées d'index par page	<p>Nombre (fixe ou moyen ou minimal ou maximal) d'entrées d'index au plus par page (ce nombre DOIT être borné par $(e_{max}(I(R, c))+1)/2$ et $e_{max}(I(R, c))$ pour les feuilles et par 1 et $e_{max}(I(R, c))$ pour la racine si elle est le seul nœud de l'arbre !) :</p> $nbMax_{entrees/page}^{fixe moy min max}(I(R, c)) = \frac{T_{page}^{fixe}(*) - T_{idPage}^{fixe}(*)}{T_{entree}^{fixe moy min max}(I(R, c))}$
$nb_{pagesEntrees}^{min max}(I(R, c))$	pages	<p>Nombre (minimal ou maximal) de pages nécessaires au stockage des entrées d'index (uniquement) :</p> <ul style="list-style-type: none"> • Si l'arbre est de hauteur maximale : $nb_{pagesEntrees}^{max}(I(R, c)) = \frac{nb_{entrees}^{exact}(I(R, c))}{\left(\frac{e_{max}(I(R, c)) + 1}{2}\right)}$ <ul style="list-style-type: none"> • Si l'arbre est de hauteur minimale : $nb_{pagesEntrees}^{min}(I(R, c)) = \frac{nb_{entrees}^{exact}(I(R, c))}{e_{max}(I(R, c))}$
$T_{info}^{fixe moy min max}(I(R, c))$	octets	<p>Taille (fixe ou moyenne ou minimale ou maximale) d'une information de circulation dans l'arbre B+ :</p> $T_{info}^{fixe moy min max}(I(R, c)) = T_{valAtt}^{fixe moy min max}(R, c) + T_{idPage}^{fixe}(*)$
$nbMax_{infos/page}^{fixe moy min max}(I(R, c))$	Informations de circulation par page	<p>Nombre (fixe ou moyen ou minimal ou maximal) d'entrées d'informations de circulation au plus par page (ce nombre DOIT être borné par $(e_{max}(I(R, c))+1)/2$ et $e_{max}(I(R, c))$ pour les nœuds non-terminaux et par 2 et $e_{max}(I(R, c))$ pour la racine !) :</p> $Nb_{infos/page}^{fixe moy min max}(I(R, c)) = \frac{T_{page}^{fixe}(*)}{T_{info}^{fixe moy min max}(I(R, c))}$

10.2.3.4.3.2 Paramètres liés aux performances d'un arbre B+

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$h_{max}(I(R, c))$	-	Hauteur de l'arbre B+ dans le pire des cas : $h_{max}(I(R, c)) \triangleq \left(\frac{\log \left(\frac{N}{2} \right)}{\log \left(\frac{e_{max}(I(R, c)) + 1}{2} \right)} \right)$
$h_{min}(I(R, c))$	-	Hauteur de l'arbre B+ dans le meilleur des cas : $h_{min} \triangleq \left(\frac{\log N}{\log e_{max}(I(R, c))} \right) - 1$
$Coût_{usage}^{min max}(I(R, c))$	Transferts de pages (en lecture)	Coût d'usage de l'index en arbre B+ : $Coût_{usage}^{min max}(I(R, c)) = h_{min max}(I(R, c)) + 1$

10.2.3.4.4 Paramètres spécifiques aux IMCAHSaR

10.2.3.4.4.1 Paramètres liés au contenu des IMCAHSaR

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$h(x)$ et N	-	Fonction de hachage telle que $0 \leq h(x) < N$
$T_{caseRepHS}^{fixe}(I(R, c))$	octets	Taille (forcément fixe) d'une case du répertoire de hachage statique : $T_{caseRepHS}^{fixe}(I(R, c)) = T_{idPage}^{fixe}(*)$
$nb_{casesRepHS}^{fixe}(I(R, c))$	octets	Nombre (forcément fixe) de cases dans le répertoire de hachage statique : $nb_{casesRepHS}^{fixe}(I(R, c)) = N$

10.2.3.4.4.2 Paramètres liés aux performances d'un IMCAHSaR

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$nbMax_{entrees/page}^{fixe moy min max}(I(R, c))$	entrées d'index par page	<p>Nombre (fixe ou moyen ou minimal ou maximal) d'entrées d'index que l'on peut stocker au plus dans une page d'un IMCAHSaR :</p> $nbMax_{entrees/page}^{fixe moy min max}(I(R, c)) \leq \frac{T_{page}^{fixe} (*) - T_{idPage}^{fixe}}{T_{entree}^{fixe moy min max}(I(R, c))}$
$nb_{pagesEntrees}^{fixe moy min max}(I(R, c))$	pages	<p>Nombre (fixe ou moyen ou minimal ou maximal) de pages contenant les entrées d'index :</p> $nb_{pagesEntrees}^{fixe moy min max}(I(R, c)) = \frac{nb_{entrees}^{exact}(I(R, c))}{nbMax_{entrees/page}^{fixe moy min max}(I(R, c))}$
$nbMax_{casesRepHS/page}^{fixe}(I(R, c))$	cases par page	<p>Nombre maximal (forcément fixe) de cases du répertoire de hachage statique au plus par page :</p> $nbMax_{casesRepHS/page}^{fixe}(I(R, c)) \leq \frac{T_{page}^{fixe} (*)}{T_{caseRepHS}^{fixe}(I(R, c))}$
$nb_{PagesRepHS}^{fixe}(I(R, c))$	pages	<p>Nombre (forcément fixe) de pages occupées par le répertoire de hachage statique :</p> $nb_{PagesRepHS}^{fixe}(I(R, c)) = \frac{nb_{casesRepHS}^{fixe}(I(R, c))}{nb_{casesRepHS/page}^{fixe}(I(R, c))}$
$T_{listePages}^{moy}(I(R, c))$	pages	<p>Taille (moyenne) des listes chaînées de pages dans lesquelles les entrées d'index sont stockées :</p> $T_{listePages}^{moy}(I(R, c)) = \frac{nb_{pagesEntrees}^{fixe moy min max}(I(R, c))}{N}$
$Coût_{usage}^{moy}(I(R, c))$	transferts de pages (en lecture)	<p>Coût (moyen) d'usage de l'IMCAHSaR :</p> $Coût_{usage}^{moy}(I(R, c)) = T_{listePages}^{moy}(I(R, c)) \underbrace{+1}_{\substack{\text{si repHS} \\ \text{non punaisé}}}$

10.2.3.4.5 Paramètres spécifiques aux IMCAHSsR

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$h(x)$ et N	-	Fonction de hachage telle que $0 \leq h(x) < N$
$nbMax_{entrees/page}^{fixe moy min max}(I(R, c))$	entrées d'index par page	Nombre (fixe ou moyen ou minimal ou maximal) d'entrées d'index que l'on peut stocker au plus dans une page d'un IMCAHSaR : $nbMax_{entrees/page}^{fixe moy min max}(I(R, c)) = \frac{T_{page}^{fixe} (*) - T_{idPage}^{fixe}}{T_{entree}^{fixe moy min max}(I(R, c))}$
$nb_{pagesEntrees}^{fixe moy min max}(I(R, c))$	pages	Nombre (fixe ou moyen ou minimal ou maximal) de pages contenant les entrées d'index : $nb_{pagesEntrees}^{fixe moy min max}(I(R, c)) = \frac{nb_{entrees}^{exact}(I(R, c))}{nbMax_{entrees/page}^{fixe moy min max}(I(R, c))}$
$T_{listePages}^{moy}(I(R, c))$	pages	Taille (moyenne) des listes chaînées de pages dans lesquelles les entrées d'index sont stockées : $T_{listePages}^{moy}(I(R, c)) = \frac{nb_{pagesEntrees}^{fixe moy min max}(I(R, c))}{N}$
$Coût_{usage}^{moy}(I(R, c))$	transferts de pages (en lecture)	Coût (moyen) d'usage de l'IMCAHSaR : $Coût_{usage}^{moy}(I(R, c)) = T_{listePages}^{moy}(I(R, c))$

10.2.3.4.6 Paramètres spécifiques aux IMCAHD

Notation	Unité(s) usuelle(s)	Paramètre (fourni ou calculé)
$h(x)$ et N	-	Fonction de hachage telle que $0 \leq h(x) < N$
$nbMax_{entrees/page}^{fixe moy min max}(I(R, c))$	entrées d'index par page	Nombre (fixe ou moyen ou minimal ou maximal) d'entrées d'index que l'on peut stocker au plus dans une page d'un IMCAHD : $nbMax_{entrees/page}^{fixe moy min max}(I(R, c)) = \frac{T_{page}^{fixe}(*)}{T_{entree}^{fixe moy min max}(I(R, c))}$
$nb_{pagesEntrees}^{fixe moy min max}(I(R, c))$	pages	Nombre (fixe ou moyen ou minimal ou maximal) de pages contenant les entrées d'index : $nb_{pagesEntrees}^{fixe moy min max}(I(R, c)) = \frac{nb_{entrees}^{exact}(I(R, c))}{nbMax_{entrees/page}^{fixe moy min max}(I(R, c))}$
$T_{caseRepHD}^{fixe}(I(R, c))$	octets	Taille (forcément fixe) d'une case du répertoire de hachage dynamique : $T_{caseRepHD}^{fixe}(I(R, c)) = T_{idPage}^{fixe}(*)$
d	cases	Profondeur du répertoire : sert à déterminer le nombre de cases du répertoire de hachage dynamique ainsi que les pseudo-clés (d bits de poids fort des codes hachés).
$nb_{casesRepHD}^{exact}(I(R, c))$	octets	Nombre (forcément exact) de cases dans le répertoire de hachage dynamique : $nb_{casesRepHD}^{exact}(I(R, c)) = 2^d$
$nbMax_{casesRepHD/page}^{exact}(I(R, c))$	cases par page	Nombre maximal (forcément exact) de cases du répertoire de hachage dynamique au plus par page : $nbMax_{casesRepHD/page}^{exact}(I(R, c)) = \frac{T_{page}^{fixe}(*)}{T_{caseRepHD}^{fixe}(I(R, c))}$
$nb_{pagesRepHD}^{exact}(I(R, c))$	pages	Nombre (forcément exact) de pages occupées par le répertoire de hachage dynamique : $nb_{pagesRepHD}^{exact}(I(R, c)) = \frac{nb_{casesRepHD}^{exact}(I(R, c))}{nb_{casesRepHD/page}^{exact}(I(R, c))}$
$Coût_{usage}^{moy}(I(R, c))$	transferts de pages (en lecture)	Coût (moyen) d'usage de l'IMCAHD : $Coût_{usage}^{moy}(I(R, c)) = 1 \quad \begin{matrix} +1 \\ \text{si } repHD \\ \text{non punaisé} \end{matrix}$

10.2.3.5 Paramètres liés à l'optimisation

Notation	Unité(s) usuelle(s)	Paramètre
-	-	Stratégie de propagation des index au-delà des opérateurs relationnels : pas de propagation, seulement quelques opérateurs relationnels (ils sont alors indiqués) ou tous les opérateurs relationnels.
-	-	Possibilité d'évaluation en pipeline : non ou oui (le nombre de pipelines simultanés au maximum est alors indiqué).
-	-	Introduction des relations dans les arbres de requête initiaux (quand cela est possible, étant donné qu'il faut tout de même respecter les jointures faites dans ladite requête !) : par ordre croissant de leur cardinalité, par ordre d'apparition dans la requête, ...
-	-	Heuristiques supportées (et indication de leur caractère optionnel/obligatoire) : <ul style="list-style-type: none"> • ALGJ : non, optionnellement (gauche et/ou droite), obligatoirement (gauche et/ou droite), • DSP : non, optionnellement, obligatoirement.
-	-	Contraintes éventuelles sur le type d'arbres de requêtes construits (initiaux ET/OU équivalents) : liste de ces contraintes particulières éventuelles...

11 Bibliographie

- Cours de M. Jacques Le Maitre, université du Sud Toulon-Var (<http://lemaître.univ-tln.fr>)
- C. J. Date, Introduction aux bases de données, Vuibert, 2001
- P. Delmal, *SQL2 – SQL3, Applications à ORACLE*, De Boeck Université, 2001
- H. Garcia-Molina, J. D. Ullman, J. Widom, *Database Systems: the Complete Book*, Computer Science Press, 2002
- G. Gardarin, Bases de données objet & relationnelles, Eyrolles, Paris, 1999
- P. M. Lewis, A. Bernstein, M. Kifer, *Databases and Transaction Processing. An Application-Oriented Approach*, Addison-Wesley, 2002

Table des illustrations

PRÉSENTATION

CHAPITRE 1 : INTRODUCTION GÉNÉRALE

Figure 1. SGBD et BD.....	4
Figure 2. Concepts basiques du paradigme relationnel	7
Figure 3. Architecture ANSI/SPARC à 3 niveaux	10
Figure 4. Schéma conceptuel relationnel.....	13
Figure 5. Instance de schéma conceptuel relationnel.....	13
Figure 6. Exemple de schéma conceptuel relationnel.....	14
Figure 7. Exemple d'instance de schéma conceptuel relationnel	14
Figure 8. Exemple de schéma conceptuel orienté objet	15
Figure 9. Exemple d'instance de schéma conceptuel orienté objet.....	15
Figure 10. Exemple d'instance de schéma conceptuel orienté objet (autre vision)	16
Figure 11. Exemple de schéma conceptuel documentaire	16
Figure 12. Exemple d'instance de schéma conceptuel documentaire	17
Figure 13. Architecture logique type d'un SGBD	19
Figure 14. Principaux mécanismes d'un SGBD abordés dans ce cours	21

LIVRE 1 : ASSURER LE CONTRÔLE DES DONNÉES

CHAPITRE 2 : INTRODUCTION AU CONTRÔLE DE DONNÉES

Figure 15. Exemple de transaction.....	26
--	----

CHAPITRE 3 : GESTIONNAIRE TRANSACTIONNEL

Figure 16. Exemple d'écriture d'une transaction	31
Figure 17. Exemple d'exécution concurrente en mise à jour immédiate	32
Figure 18. Exemple d'exécution concurrente mise à jour différée.....	33
Figure 19. Exemple de perte de mise à jour.....	35
Figure 20. Exemple de lecture impropre (données incohérentes)	36
Figure 21. Exemple de lecture impropre (données non confirmées)	37
Figure 22. Exemple de lecture non-reproductible.....	37
Figure 23. Réécriture sans problème de l'exemple de perte de mise à jour	38
Figure 24. Exemple d'équivalence : exécution concurrente E_1	40
Figure 25. Exemple d'équivalence : exécution en série E_2 équivalente à E_1	41
Figure 26. Exemple de non-équivalence : exécution en série E_3 non-équivalente à E_2 ni E_1	42
Figure 27. Exemple de graphe de précédence	48
Figure 28. Rappel de l'exemple de graphe de précédence	49
Figure 29. Autre exemple de graphe de précédence	50
Figure 30. Graphe de précédence de E_1 ne contenant aucun cycle.....	51
Figure 31. Graphe de précédence de E_2 contenant un cycle	51
Figure 32. Rappel de l'exemple de perte de mise à jour	52
Figure 33. Graphe de précédence de l'exemple de perte de mise à jour	52
Figure 34. Rappel de l'exemple de lecture impropre (données incohérentes)	52
Figure 35. Graphe de précédence de l'exemple de données incohérentes	53
Figure 36. Rappel de l'exemple de lecture impropre (données non confirmées)	53
Figure 37. Graphe de précédence de l'exemple de données non confirmées	53
Figure 38. Rappel de l'exemple de lecture non-reproductible	53
Figure 39. Graphe de précédence de l'exemple de lecture non reproductible	54
Figure 40. Exemple d'utilisation des événements de verrouillage/déverrouillage.....	56
Figure 41. Exemple de verrouillage dégradant les performances	57
Figure 42. Exemple de lecture non-reproductible malgré les verrouillages	58
Figure 43. Exemple de transaction à 2 phases	59
Figure 44. Exemple de perte de mise à jour réécrit avec des T2P	61
Figure 45. Graphe de précédence de l'exemple de perte de mise à jour (T2P).....	61
Figure 46. Exemple de lecture impropre (données incohérentes) réécrit avec des T2P	61
Figure 47. Graphe de précédence de l'exemple de lecture impropre (données incohérentes) (T2P).....	62
Figure 48. Exemple de lecture impropre (données non confirmées) réécrit avec des T2P	62
Figure 49. Graphe de précédence de l'exemple lecture impropre (données non confirmées) (T2P).....	62
Figure 50. Exemple de lecture non reproductible réécrit avec des T2P	62
Figure 51. Graphe de précédence de l'exemple lecture non reproductible (T2P).....	63
Figure 52. Exemple d'interblocage.....	63
Figure 53. Exemple de graphe d'attente montrant un interblocage.....	64
Figure 54. Graphe d'attente de la perte de mise à jour réécrite en T2P	65
Figure 55. Graphe d'attente de la lecture impropre de données incohérentes réécrite en T2P	65
Figure 56. Graphe d'attente de la lecture impropre de données non confirmées réécrite en T2P	65
Figure 57. Graphe d'attente de la lecture non reproductible réécrite en T2P	65

Figure 58. Exemple de transaction à deux phases amoindrissant les accès concurrentiels	66
Figure 59. Exemple d'utilisation des opérations de reclassement	68
Figure 60. Niveaux de granularité « théoriques » dans une BD	69
Figure 61. Exemple de problème d'objet fantôme.....	72
Figure 62. Exemple de problème d'objet fantôme résolu grâce au verrouillage intentionnel	73

CHAPITRE 4 : GESTIONNAIRE DE REPRISE APRÈS PANNE

Figure 63. Exemple de panne pendant une exécution concurrente.....	78
Figure 64. Exemple de panne pendant une exécution avec accès concurrentiels.....	80
Figure 65. Exemple de tenue de journal de reprise après panne	83
Figure 66. Exemple d'exécution concurrente à journaliser	87
Figure 67. Exemple de journal de reprise (en mode de mise à jour immédiate).....	87
Figure 68. Exemple de journal de reprise (en mode de mise à jour différée)	88

LIVRE 2 : OFFRIR DES PERFORMANCES OPTIMALES

CHAPITRE 5 : INTRODUCTION À L'OPTIMISATION

Figure 69. Organisation d'un disque en plateaux.....	92
Figure 70. Têtes de lecture/écriture d'un disque	93
Figure 71. Organisation en pistes d'un disque	94
Figure 72. Organisation en cylindres d'un disque	94
Figure 73. Organisation « en quartiers » d'un plateau.....	95
Figure 74. Organisation en secteurs d'un plateau.....	95
Figure 75. Secteurs (physiques) et blocs (logiques, OS)	97
Figure 76. Taille « brute » vs taille occupée sur disque d'un fichier (Windows 10 1903)	98
Figure 77. Transferts de données mémoire de stockage ↔ mémoire de travail	100
Figure 78. Transferts de blocs mémoire de stockage ↔ mémoire de travail.....	100
Figure 79. Secteurs (physiques), blocs (logiques, OS) et pages (logiques, OS)	102
Figure 80. Accès page par page aux données contenues dans un bloc	103
Figure 81. Contenu (grossier) d'une BD	107
Figure 82. Organisation des pages de stockage des données d'une BD	107
Figure 83. Processus BPMN de traitement d'une requête	108
Figure 84. Dispositifs d'optimisation de l'évaluation de requêtes	108

CHAPITRE 6 : ORGANISATION PHYSIQUE

Figure 85. Organisation classique d'une page p de stockage de n-uplets	111
Figure 86. Modèle d'organisation d'une page de stockage des n-uplets	112
Figure 87. Déplacement absolu	113
Figure 88. Déplacements relatifs (en avant ou en arrière)	113
Figure 89. Données d'un n-uplet.....	116
Figure 90. Stockage contigu des données (totales) de n-uplets courts	119
Figure 91. Stockage « discontinu » de n-uplets au sein d'une page	119
Figure 92. Stockage « contigu » des données (totales) des n-uplets longs à attributs courts	120
Figure 93. Stockage « contigu » des données (totales) des n-uplets longs à attribut(s) long(s)	121
Figure 94. Réduction de la taille d'un n-uplet	122
Figure 95. Agrandissement de la taille d'un n-uplet au sein d'une page pouvant toujours l'accueillir	123
Figure 96. Adressage physique vs adressage logique de n-uplets	125
Figure 97. Pages de stockage d'une relation en mode d'adressage direct	126
Figure 98. Mise en œuvre de pointeurs de suivi en mode d'adressage direct	127
Figure 99. Contenu d'une table de correspondance	129
Figure 100. Pages de stockage d'une relation en mode d'adressage indirect	131
Figure 101. Pages d'une table de correspondance vs pages de n-uplets pour une relation R	132

CHAPITRE 7 : GESTIONNAIRE DE MÉMOIRE CACHE

Figure 102. Demande d'accès en lecture à une page p SANS mémoire cache	147
Figure 103. Demande d'accès en écriture à une page p SANS mémoire cache.....	148
Figure 104. Structure et contenu de la mémoire cache	149
Figure 105. Demande d'accès en lecture à une page p AVEC mémoire cache	150
Figure 106. Demande d'accès en écriture à une page p AVEC mémoire cache.....	152
Figure 107. Demande de vidage (total ou, ici, partiel) de la mémoire cache	153

CHAPITRE 8 : INDEXATION DES DONNÉES

Figure 108. Répartition ordonnée des n-uplets quand R est indexée par un index groupé	163
Figure 109. Répartition équiprobable des n-uplets quand R est indexée par un index non-groupé	164
Figure 110. Diagramme de classes UML modélisant le contenu d'un index	167
Figure 111. Lien entre pages de l'index et de la relation indexée en mode d'adressage direct	167
Figure 112. Lien entre pages de l'index et de la relation indexée en mode d'adressage indirect	168
Figure 113. Exemple d'arbre B	173

Figure 114. Exemple d'arbre B+	173
Figure 115. Exemple de feuilles d'un arbre B+ dont l'ordre $e_{max}(l(R, c))$ vaut 3	176
Figure 116. Allure générale d'un arbre B+	178
Figure 117. Pages de stockage d'un arbre B+ et de la relation indexée (adressage direct)	180
Figure 118. Pages de stockage d'un arbre B+ et de la relation indexée (adressage indirect)	181
Figure 119. Illustration (grossière) de l'objectif de la technique du hachage	206
Figure 120. Cas limite de « mauvais hachage » (tailles des partitions très inégales)	207
Figure 121. Cas limite de « mauvais hachage » (code hachés inutilisés)	208
Figure 122. Exemple d'index mono-critère à accès par hachage statique avec répertoire	213
Figure 123. Répartition des entrées d'index (v, a) en fonction de leur code haché $h(v)$	213
Figure 124. Pages de stockage d'un IMCAHSaR et de la relation indexée (adressage direct)	214
Figure 125. Pages de stockage d'un IMCAHSaR et de la relation indexée (adressage indirect)	215
Figure 126. Exemple d'index mono-critère à accès par hachage statique avec répertoire	227
Figure 127. Exemple d'index mono-critère à accès par hachage statique sans répertoire	228
Figure 128. Pages de stockage d'un IMCAHSsR et de la relation indexée (adressage direct)	229
Figure 129. Pages de stockage d'un IMCAHSsR et de la relation indexée (adressage indirect)	230
Figure 130. Organisation générale d'un index mono-critère à accès par hachage dynamique	233
Figure 131. Pages de stockage d'un IMCAHD et de la relation indexée (adressage direct)	235
Figure 132. Pages de stockage d'un IMCAHD et de la relation indexée (adressage indirect)	236
Figure 133. Mauvais et bon résultat du doublement du répertoire de hachage dynamique	240

CHAPITRE 9 : OPTIMISATION DE REQUÊTES

Figure 134. Deux niveaux d'optimisation d'une requête	251
Figure 135. Optimisation <i>a priori</i>	253
Figure 136. Explosion combinatoire (requête \equiv arbre initial / arbres équivalents / plans d'exécution)	253
Figure 137. Exemple d'arbre de requête.....	254
Figure 138. Exemple d'arbre de requête initial	257
Figure 139. Transformation T_1 de regroupement/éclatement/permotion des sélections	258
Figure 140. Transformation T_2 de permutation sélection \leftrightarrow projection	259
Figure 141. Transformation T_3 de permutation sélection \leftrightarrow jointure	261
Figure 142. Transformation T_4 de permutation jointure \leftrightarrow projection	262
Figure 143. Transformation T_5 de commutativité de la jointure	262
Figure 144. Transformation T_6 d'associativité de la jointure	263
Figure 145. Forme générale d'un arbre linéaire gauche de jointure	265
Figure 146. Arbres linéaires gauches et droits de jointure	266
Figure 147. Arbre de requête résultant de l'application de l'heuristique DSP	268
Figure 148. Méthodes de résolution et plans d'exécution d'un arbre de requête	271
Figure 149. Répartition ordonnée des n-uplets quand R est indexée par un index groupé	278
Figure 150. Répartition équiprobable des n-uplets quand R est indexée par un index non-groupé	278
Figure 151. Optimisation (étude de cas) : arbre de requête initial A_{11}	301
Figure 152. Optimisation (étude de cas) : arbre de requête équivalent A_{12}	301
Figure 153. Optimisation (étude de cas) : arbres de requête équivalents A_{21} et A_{22}	302
Figure 154. Optimisation (étude de cas) : arbres de requête équivalents A_{31} et A_{32}	302
Figure 155. Optimisation (étude de cas) : arbres de requête équivalents A_{41} et A_{42}	303
Figure 156. Étiquetage d'une sélection	304
Figure 157. Étiquetage d'une jointure	305
Figure 158. Étiquetage d'une projection	305
Figure 159. Optimisation (étude de cas) : étiquetage des relations de l'arbre de requête A_{11}	308
Figure 160. Optimisation (étude de cas) : étiquetage complet de l'arbre de requête A_{11}	309
Figure 161. Optimisation (étude de cas) : étiquetage complet de l'arbre de requête A_{21}	315
Figure 162. Optimisation (étude de cas) : étiquetage complet de l'arbre de requête A_{31}	315
Figure 163. Optimisation (étude de cas) : étiquetage complet de l'arbre de requête A_{41}	316
Figure 164. Optimisation (étude de cas) : application de l'heuristique DSP à l'arbre de requête A_{41}	317
Figure 165. Transmission d'une relation temporaire sans pipeline	321
Figure 166. Transmission d'une relation temporaire avec pipeline	321
Figure 167. Niveaux dans l'arbre de requête A_{51} pour les évaluations en pipeline	323
Figure 168. Optimisation (étude de cas) : prise en compte d'un pipeline pour l'arbre A_{11}	324
Figure 169. Optimisation (étude de cas) : prise en compte d'un pipeline pour l'arbre A_{21}	325
Figure 170. Optimisation (étude de cas) : prise en compte d'un pipeline pour l'arbre A_{31}	325
Figure 171. Optimisation (étude de cas) : prise en compte d'un pipeline pour l'arbre A_{41} (cas 1)	326
Figure 172. Optimisation (étude de cas) : prise en compte d'un pipeline pour l'arbre A_{41} (cas 2)	326
Figure 173. Optimisation (étude de cas) : prise en compte de 2 pipelines pour l'arbre A_{41}	327

SYNTHESE

CHAPITRE 10 : RÉCAPITULATIF DE L'ENSEMBLE DES PARAMÈTRES

Aucune entrée de table d'illustration n'a été trouvée.

Table des tableaux

PRÉSENTATION

CHAPITRE 1 : INTRODUCTION GÉNÉRALE

Tableau 1. Apports des principaux mécanismes d'un SGBD	22
--	----

LIVRE 1 : ASSURER LE CONTRÔLE DES DONNÉES

CHAPITRE 2 : INTRODUCTION AU CONTRÔLE DES DONNÉES

Aucune entrée de table d'illustration n'a été trouvée.

CHAPITRE 3 : GESTIONNAIRE TRANSACTIONNEL

Tableau 2. Événements transactionnels basiques.....	31
Tableau 3. Nature des opérations transactionnelles selon le mode de mise à jour	43
Tableau 4. Permutation de 2 opérations op_{NoBD} sans accès BD	44
Tableau 5. Permutation d'une opération op_{NoBD} sans accès BD et d'une lecture op_{Lect}	44
Tableau 6. Permutation d'une opération op_{NoBD} sans accès BD et d'une écriture op_{Ecrit}	45
Tableau 7. Permutation de 2 lectures op_{Lect} de 2 données différentes	45
Tableau 8. Permutation de 2 lectures op_{Lect} d'une même donnée.....	45
Tableau 9. Permutation d'une lecture op_{Lect} et d'une écriture op_{Ecrit} de 2 données différentes.....	45
Tableau 10. Permutation d'une lecture op_{Lect} et d'une écriture op_{Ecrit} d'une même donnée	46
Tableau 11. Permutation de 2 écritures op_{Ecrit} de 2 données différentes.....	46
Tableau 12. Permutation de 2 écritures op_{Ecrit} d'une même donnée	46
Tableau 13. Synthèse des conflictualités entre types d'opérations transactionnelles	46
Tableau 14. Matrice de compatibilité des verrous en mode S et X	55
Tableau 15. Événements transactionnels de verrouillage/déverrouillage	56
Tableau 16. Avantages et inconvénients des stratégies de traitement de l'interblocage	66
Tableau 17. Événements transactionnels de reclassement	67
Tableau 18. Matrice de compatibilité des verrous « classiques » ET intentionnels	71
Tableau 19. Paramètres du gestionnaire transactionnel	74

CHAPITRE 4 : GESTIONNAIRE DE REPRISE APRÈS PANNE

Tableau 20. Événements consignés dans le journal de reprise	83
---	----

LIVRE 2 : OFFRIR DES PERFORMANCES OPTIMALES

CHAPITRE 5 : INTRODUCTION À L'OPTIMISATION

Tableau 21. Paramètres de la mémoire de stockage du SGBD (au niveau physique).....	96
Tableau 22. Paramètres de la mémoire de stockage du SGBD (au niveau des blocs logiques)	99
Tableau 23. Paramètres de la mémoire (de stockage et de travail) du SGBD (au niveau des pages logiques).....	104
Tableau 24. Opérations en mémoire (stockage et travail) pour la gestion des données	105

CHAPITRE 6 : ORGANISATION PHYSIQUE

Tableau 25. Bilan des stratégies d'adressage des n-uplets	133
Tableau 26. Bilan des stratégies de stockage des valeurs d'attributs	134
Tableau 27. Bilan des stratégies de gestion du répertoire des déplacements	136
Tableau 28. Performances d'accès aux données : paramètres SGBD (génériques).....	141
Tableau 29. Performances d'accès aux données : paramètres SGBD (n-uplets d'une relation R)	142
Tableau 30. Performances d'accès aux données : paramètres SGBD (PS ou TC d'une relation R)	143
Tableau 31. Performances d'accès aux données : nombre « d'informations » par page (d'une relation R)	144
Tableau 32. Performances d'accès aux données : nombre de pages occupées (d'une relation R).....	145

CHAPITRE 7 : GESTIONNAIRE DE MÉMOIRE CACHE

Tableau 33. Paramètres du gestionnaire de mémoire cache	158
---	-----

CHAPITRE 8 : INDEXATION DES DONNÉES

Tableau 34. Performances d'indexation : paramètres des n-uplets de la relation indexée	169
Tableau 35. Performances d'indexation : paramètres généraux (indépendants de sa structure) de l'index	170
Tableau 36. Structures courantes d'index.....	171
Tableau 37. Relation Dictionnaire illustrant les structures d'index présentées.....	172
Tableau 38. Raisonnement sur le contenu d'un arbre B+ (pire des cas)	199
Tableau 39. Raisonnement sur le nombre de noeuds occupés par un arbre B+ (pire des cas)	200
Tableau 40. Raisonnement sur le contenu d'un arbre B+ (meilleur des cas)	202
Tableau 41. Raisonnement sur le nombre de noeuds occupés par un arbre B+ (meilleur des cas)	203
Tableau 42. Performances d'indexation : paramètres spécifiques au contenu des arbres B+	204
Tableau 43. Performances d'indexation : paramètres spécifiques aux performances des arbres B+	205

Tableau 44. Codes hachés utilisés pour l'exemple d'IMCAHSaR	219
Tableau 45. Performances d'indexation : paramètres spécifiques aux IMCAHSaR	226
Tableau 46. Bilan des organisations d'IMCAHS	231
Tableau 47. Performances d'indexation : paramètres spécifiques aux IMCAHSaR	231
Tableau 48. Bits de poids fort des codes hachés utilisés pour l'exemple d'IMCAHD	241
Tableau 49. Performances d'indexation : paramètres spécifiques aux IMCAHD.....	247

CHAPITRE 9 : OPTIMISATION DE REQUÊTES

Tableau 50. Exemples de nombres d'ordres linéaires de jointures sur n relations	264
Tableau 51. Comparaison des nombres d'ordres linéaires simples et gauches de jointures	266
Tableau 52. Plans d'exécution et coûts.....	271
Tableau 53. Paramètres généraux (machine, OS, SGBD)	274
Tableau 54. Paramètres relatifs aux constituants/attributs d'une relation R	275
Tableau 55. Paramètres relatifs aux n-uplets d'une relation R	276
Tableau 56. Paramètres relatifs à la globalité d'une relation R	277
Tableau 57. Paramètres relatifs à un index I	277
Tableau 58. Avantages et inconvénients de la propagation des index.....	279
Tableau 59. Paramètres spécifiques à une requête	280
Tableau 60. Notation et portion d'arbre de requête associée à une sélection	281
Tableau 61. Notation et portion d'arbre de requête associée à une jointure.....	285
Tableau 62. Notation et portion d'arbre de requête associée à une projection	290
Tableau 63. Bilan de la mise en œuvre d'1 ou 2 pipelines sur l'étude de cas.....	328
Tableau 64. Paramètres liés à l'optimisation de requêtes	328

SYNTHÈSE

CHAPITRE 10 : RÉCAPITULATIF DE L'ENSEMBLE DES PARAMÈTRES

Aucune entrée de table d'illustration n'a été trouvée.

Table des équations

PRÉSENTATION

CHAPITRE 1 : INTRODUCTION GÉNÉRALE

Aucune entrée de table d'illustration n'a été trouvée.

LIVRE 1 : ASSURER LE CONTRÔLE DES DONNÉES

CHAPITRE 2 : INTRODUCTION AU CONTRÔLE DES DONNÉES

Aucune entrée de table d'illustration n'a été trouvée.

CHAPITRE 3 : GESTIONNAIRE TRANSACTIONNEL

Équation 1. Nombre de possibilités d'exécutions en série de n transactions 39

CHAPITRE 4 : GESTIONNAIRE DE REPRISE APRÈS PANNE

Aucune entrée de table d'illustration n'a été trouvée.

LIVRE 2 : OFFRIR DES PERFORMANCES OPTIMALES

CHAPITRE 5 : INTRODUCTION À L'OPTIMISATION

Équation 2. Lien entre taille d'un bloc et taille d'un secteur 97

Équation 3. Liens entre tailles de page, bloc et secteur 102

CHAPITRE 6 : ORGANISATION PHYSIQUE

Équation 4. Taille totale d'un n-uplet 117

Équation 5. Taille des données (brutes) d'un n-uplet 117

Équation 6. Taille totale d'un pointeur de suivi 128

Équation 7. Taille (fixe) des données (brutes) d'un pointeur de suivi 128

Équation 8. Nombre de lignes dans une table de correspondance 131

Équation 9. Taille (fixe) d'une ligne d'une table de correspondance 131

Équation 10. Taille (totale) d'un n-uplet d'une relation R 138

Équation 11. Adressage direct : impact sur les métadonnées des n-uplets 138

Équation 12. Adressage direct : taille (totale) d'un pointeur de suivi 138

Équation 13. Adressage direct : taille des données (brutes) d'un pointeur de suivi 138

Équation 14. Adressage direct : taille des métadonnées des pointeurs de suivi 138

Équation 15. Bornage (trivial) du nombre de pointeurs de suivi dans une relation R 138

Équation 16. Taille (fixe) d'une ligne d'une table de correspondance 139

Équation 17. Nombre de lignes dans une table de correspondance 139

Équation 18. Format fixe : taille des données (brutes) des n-uplets 139

Équation 19. Format variable : taille des données (brutes) des n-uplets 139

Équation 20. Format variable : impact sur les métadonnées des n-uplets 139

Équation 21. Gestion statique : impact sur la place « utile » restant dans une page 140

Équation 22. Gestion dynamique : aucun impact sur la place « utile » dans une page 140

Équation 23. Gestion dynamique : impact sur les métadonnées des n-uplets 140

Équation 24. Gestion dynamique (Si adressage direct) : impact sur les métadonnées des pointeurs de suivi 140

CHAPITRE 7 : GESTIONNAIRE DE MÉMOIRE CACHE

Aucune entrée de table d'illustration n'a été trouvée.

CHAPITRE 8 : INDEXATION DES DONNÉES

Équation 25. Coût d'une recherche « brute » des n-uplets tel que $c = v$ 161

Équation 26. Coût d'une recherche indexée des n-uplets tels que $c = v$ 162

Équation 27. Coût du chargement des n-uplets $c = v$ quand l'index $I/R, c$ est groupé 163

Équation 28. Coût du chargement des n-uplets $c = v$ quand l'index $I/R, c$ est non-groupé 164

Équation 29. Nombre d'entrées d'index dans un index primaire 166

Équation 30. Taille de la liste d'adresses logiques des entrées d'un index primaire 166

Équation 31. Nombre d'entrées d'index dans un index secondaire 166

Équation 32. Taille de la liste d'adresses logiques des entrées d'un index secondaire 166

Équation 33. Remplissage des feuilles (hors racine) d'un arbre B+ 176

Équation 34. Remplissage des nœuds non-terminaux (hors racine) d'un arbre B+ 177

Équation 35. Remplissage de la racine d'un arbre B+ (quand elle est son seul nœud) 179

Équation 36. Remplissage de la racine d'un arbre B+ (quand elle n'est pas son seul nœud) 179

Équation 37. Nombre total d'enregistrements d'index au niveau n d'un arbre B+ (pire des cas) 198

Équation 38. Nombre d'entrées d'index dans les feuilles d'un arbre B+ (pire des cas) 199

Équation 39. Niveaux et hauteur d'un arbre B+ pour y stocker N entrées d'index (pire des cas) 199

Équation 40. Nombre total d'enregistrements d'index au niveau n d'un arbre B+ (meilleur des cas) 201

Équation 41. Nombre d'entrées d'index dans les feuilles d'un arbre B+ (meilleur des cas).....	202
Équation 42. Niveaux et hauteur d'un arbre B+ pour y stocker N entrées d'index (meilleur des cas)	202

CHAPITRE 9 : OPTIMISATION DE REQUÊTES

Équation 43. Nombre d'ordres linéaires de jointures sur n relations.....	263
Équation 44. Nombre d'ordres linéaires gauches de jointures sur n relations.....	265
Équation 45. Coût d'un plan d'exécution.....	271
Équation 46. Coût d'un opérateur relationnel (mis en œuvre par une méthode donnée)	273
Équation 47. Coût du chargement des n-uplets $c = v$ quand l'index $I(R, c)$ est groupé	278
Équation 48. Coût du chargement des n-uplets $c = v$ quand l'index $I(R, c)$ est non-groupé	278
Équation 49. Coût de production d'une sélection évaluée par un parcours séquentiel.....	282
Équation 50. Coût de production d'une sélection évaluée par un parcours indexé (I non-groupé)	284
Équation 51. Coût de production d'une sélection évaluée par un parcours indexé (I groupé).....	284
Équation 52. Coût de production d'une jointure évaluée par des boucles imbriquées	286
Équation 53. Coût de production d'une jointure évaluée par des boucles optimisées	287
Équation 54. Coût de production d'une jointure évaluée par une boucle indexée (I non-groupé).....	289
Équation 55. Coût de production d'une jointure évaluée par une boucle indexée (I groupé).....	289
Équation 56. Coût de production d'une projection évaluée par un parcours séquentiel.....	291
Équation 57. Coût de production d'une projection évaluée par un parcours indexé	292
Équation 58. Coût d'écriture de la relation résultat d'un opérateur relationnel	293
Équation 59. Évaluation de la cardinalité de la relation résultat d'une sélection	294
Équation 60. Évaluation de la taille des données des n-uplets produits par une sélection.....	295
Équation 61. Évaluation de la cardinalité de la relation résultat d'une jointure	295
Équation 62. Évaluation de la taille des données des n-uplets produits par une jointure	296
Équation 63. Évaluation de la cardinalité de la relation résultat d'une projection	296
Équation 64. Évaluation de la taille des données des n-uplets produits par une projection.....	296
Équation 65. Triplet étiquetant chaque relation R d'un arbre de requête.....	303

SYNTHÈSE

CHAPITRE 10 : RÉCAPITULATIF DE L'ENSEMBLE DES PARAMÈTRES

Équation 43. Nombre d'ordres linéaires de jointures sur n relations.....	263
Équation 44. Nombre d'ordres linéaires gauches de jointures sur n relations.....	265
Équation 45. Coût d'un plan d'exécution.....	271
Équation 46. Coût d'un opérateur relationnel (mis en œuvre par une méthode donnée)	273
Équation 47. Coût du chargement des n-uplets $c = v$ quand l'index $I(R, c)$ est groupé	278
Équation 48. Coût du chargement des n-uplets $c = v$ quand l'index $I(R, c)$ est non-groupé	278
Équation 49. Coût de production d'une sélection évaluée par un parcours séquentiel.....	282
Équation 50. Coût de production d'une sélection évaluée par un parcours indexé (I non-groupé)	284
Équation 51. Coût de production d'une sélection évaluée par un parcours indexé (I groupé).....	284
Équation 52. Coût de production d'une jointure évaluée par des boucles imbriquées	286
Équation 53. Coût de production d'une jointure évaluée par des boucles optimisées	287
Équation 54. Coût de production d'une jointure évaluée par une boucle indexée (I non-groupé).....	289
Équation 55. Coût de production d'une jointure évaluée par une boucle indexée (I groupé).....	289
Équation 56. Coût de production d'une projection évaluée par un parcours séquentiel.....	291
Équation 57. Coût de production d'une projection évaluée par un parcours indexé	292
Équation 58. Coût d'écriture de la relation résultat d'un opérateur relationnel	293
Équation 59. Évaluation de la cardinalité de la relation résultat d'une sélection	294
Équation 60. Évaluation de la taille des données des n-uplets produits par une sélection.....	295
Équation 61. Évaluation de la cardinalité de la relation résultat d'une jointure	295
Équation 62. Évaluation de la taille des données des n-uplets produits par une jointure	296
Équation 63. Évaluation de la cardinalité de la relation résultat d'une projection	296
Équation 64. Évaluation de la taille des données des n-uplets produits par une projection.....	296
Équation 65. Triplet étiquetant chaque relation R d'un arbre de requête.....	303