Université de Sherbrooke

Systèmes de gestion de bases de données

Indexation

UdeS:SGBD_02

Christina KHNAISSER *(christina.khnaisser@usherbrooke.ca)*

Luc LAVOIE *(luc.lavoie@usherbrooke.ca)*

(les auteurs sont cités en ordre alphabétique nominal)

—

CoFELI/Scriptorum/SGBD_02-Indexation (v103), version 1.0.1.a, en date du 2025-09-15

— document de travail, ne pas citer —

Sommaire

Introduction aux techniques d’indexation relationnelle utilisées par les systèmes de gestion de bases de données.

Mise en garde

Le présent document est en cours d’élaboration ; en conséquence, il est incomplet et peut contenir des erreurs.

Historique

diffusion	resp.	description
2025-09-30	LL	Revue préalable aux activités 2025-3.
2024-09-17	CK	Ébauche à partir des travaux d’Elmasri et Navathe.
2024-08-16	LL	Ébauche initiale à partir de documents antérieurs produits entre 2004 et 2023.

Table des matières

Introduction

Le présent document a pour but de :

- Présenter l'indexation du point de vue des structures de données.
- Distinguer index (sous-ensemble d'attributs indexés) et index (structure d'indexation).

Évolution du document

La première version du document a été établie sur les bases des différents travaux publiés par Elmasri.

1. Présentation

1.1. Aperçu

L'indexation est utilisée pour améliorer la performance de l'exécution de requêtes selon diverses conditions de recherche.

Lorsqu'une requête est exécutée, les index peuvent être utilisés pour extraire une liste de pointeurs vers les enregistrements dont les valeurs satisfont les conditions de recherche et seules les pages les contenant sont lues (si elles ne sont pas déjà en mémoire grâce à la pagination).

Un index est une structure de données qui fournissent des chemins d'accès secondaires selon un ensemble d'attributs. Autrement dit, c'est un moyen alternatif d'accéder aux données indépendamment de l'organisation physique des enregistrements dans les fichiers principaux sur le disque.

Les SGBD offrent une variété d'index supportant plusieurs types de données et plusieurs classes de conditions de recherche.

1.2. Motivation

Les trois opérations relationnelles primaires déterminent toutefois des besoins plus spécifiques quant à l'indexation :

- Indexation de renommage : sélection d'un identifiant parmi l'ensemble des identifiants d'une structure non scalaire (tuplet, relation de base, relation virtuelle).
- Indexation de restriction : sélection des tuplets d'une relation en fonction de la valeur de leurs attributs ; elle est primordiale non seulement pour la restriction, mais aussi pour la jointure, l'union, la différence (et toutes les opérations qui sont composées à l'aide de celles-ci, dont le produit cartésien et le regroupement).
- Indexation de projection : sélection des « colonnes » d'une relation en fonction de leur identifiant ; (une colonne est une structure constituée des valeurs d'un même attribut de chacun des tuplets d'une relation).

Les autres opérations relationnelles primaires (jointure, union, différence), de même que les autres opérations binaires internes en découlant (produit cartésien, intersection, etc.) utilisent la composition des autres formes d'indexation. Ces modes de composition seront traitées dans le module `SGBD_03-Execution-et-optimisation`.

Indexation de renommage

L'indexation de renommage est caractérisée par la faible de taille de la cible et la faible cardinalité de la cible comme de l'ensemble de recherche. Elle peut être traitée simplement sur la base d'algorithmes et de structures déjà bien étudiées en structures de données. Elle a peu d'incidence sur l'efficience SGBD. Elle n'a guère retenu l'attention spécifique des maitres d'oeuvre de SGBD.

En fait, le renommage n'intervient que dans deux contextes :

- L'évaluation d'expression, auquel cas il n'a aucune incidence sur les fichiers (autant ceux du catalogue que ceux des données).
- Le renommage d'entités du catalogue, où il est traité comme une mise à jour de données (appliquées aux variables de relation du catalogue).

Indexation de projection

L'indexation de projection est caractérisée par la taille éventuellement considérable des entités de la cible (les colonnes) bien que l'ensemble de recherche demeure en général restreint (le nombre d'attributs d'une relation). Elle a toutefois fait l'objet d'une étude approfondie afin de soutenir la mise à disposition de bases

de données dites « verticales » plus particulièrement efficientes dans le traitement de données multimétriques chronologiques. Plusieurs articles de recherche leur ont été consacrés et plusieurs produits en ont tiré parti (par exemple *Vertica* [Stonebraker, HP, MicroFocus, OpenText], *Vertical scaling* [Oracle]).

Indexation de restriction

L'indexation de restriction est caractérisée par la taille parfois variable, mais éventuellement importante, des entités de la cible (les clés de recherche) et une cardinalité variable, mais éventuellement considérable, des ensembles de recherche (le nombre de tuplets d'une relation). Elle a fait l'objet, et fait toujours l'objet, d'une étude approfondie et de très nombreux articles de recherche. Il est fréquent qu'un même SGBD offre plusieurs stratégies différentes (notamment, spécifiable par la commande INDEX en SQL). La variété des stratégies offertes et la qualité de leur mise en oeuvre sont généralement considérées comme un facteur déterminant de l'efficience des SGBD.

Le présent document, l'indexation fera donc uniquement référence à l'indexation de restriction.

2. Type d'organisation des pages d'enregistrements

2.1. Fichier non ordonné

Les enregistrements sont ajoutés au fur et à mesure de leur arrivée.

L'insertion est très efficace : un enregistrement est inséré dans le premier espace disponible. Si un enregistrement est trop long par rapport à l'espace libre, la portion qui déborde est stockée dans la page suivante.

La recherche est couteuse : une **recherche linéaire** est effectuée. Ceci nécessite en moyenne la lecture de la moitié des pages du fichier $\Theta(b/2)$.

En PostgreSQL, les données sont organisées ainsi.

2.2. Fichier ordonné

Les enregistrements sont ordonnés selon une valeur d'un champ (la clé d'ordonnancement).

L'insertion est couteuse : l'enregistrement doit être inséré au bon endroit afin de maintenir l'ordonnancement de la page.

La suppression peut causer des « trous » dans la page. Un mécanisme pour faire le « ménage » existe.

La recherche est efficace : une **recherche binaire** est effectuée en utilisant la clé d'ordonnancement. Ceci nécessite en moyenne la lecture de $\Theta(\log(n))$ de pages du fichier.

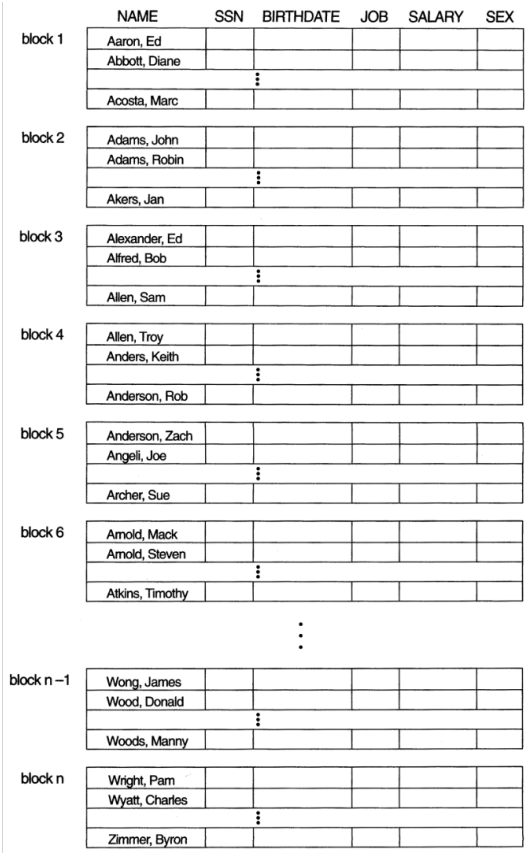

Figure 1. Illustration des pages du fichier ordonnée [Elmasri2016] p.569.

2.3. Fichier dispersé

Les enregistrements sont groupés en partitions (*buckets*) selon un champ (la clé de hachage).

- Une partition correspond à une ou plusieurs pages [virtuellement ou physiquement ?] contigües du fichier.
- Une fonction de hachage appliquée à la valeur du champ retourne l'emplacement de la partition qui contient les enregistrements ayant la valeur du champ.

L'insertion est efficace, mais avec des risques de collisions. Plusieurs mécanismes existent pour gérer les collisions : en créant un tableau connexe ou une liste.

La recherche est efficace : l'identification de la page à lire s'effectue en mémoire en temps constant $\Theta(1)$. La recherche est très efficace, surtout si elle est effectuée avec une condition d'égalité sur la clé de hachage.

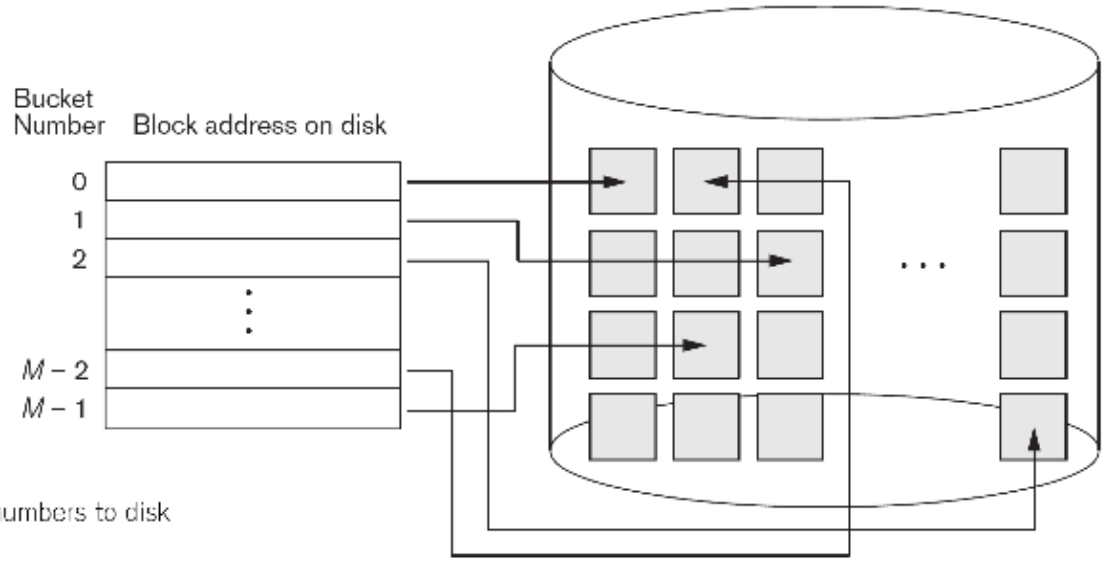

Figure 17.9
Matching bucket numbers to disk block addresses.

Figure 2. Illustration d'ensemble de partitions et la correspondance avec les pages [Elmasri2016] p.576.

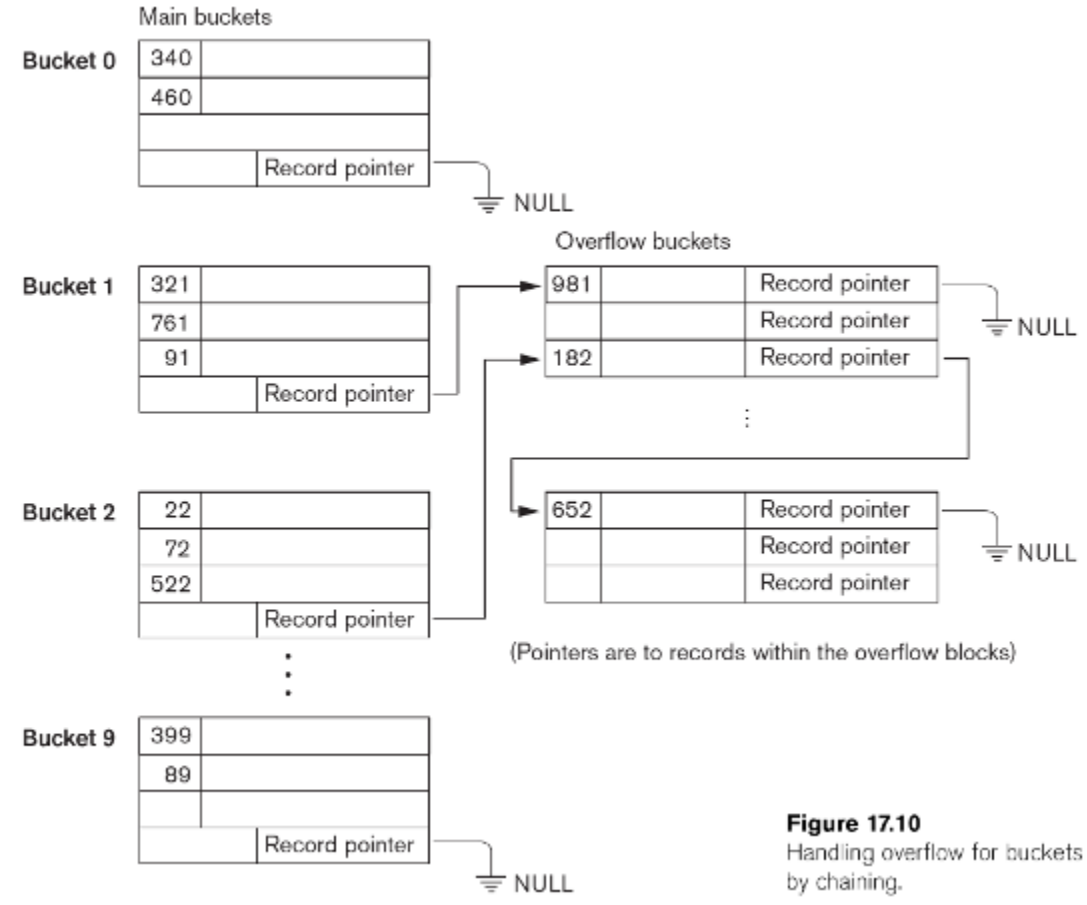

Figure 17.10
Handling overflow for buckets by chaining.

Figure 3. Illustration gestion du débordement des partitions [Elmasri2016] p.576.

2.4. Bloc et page, même combat ?

La structure de page a été introduite principalement pour deux raisons :

- s'affranchir de la variété de tailles de bloc des dispositifs de stockage lors de la gestion des fichiers ;

- profiter du gain découlant de l'accès à des blocs consécutifs (réduction de la latence et augmentation du débit) au prix d'une granularité plus forte (pouvant occasionner à l'occasion une augmentation de l'espace requis en mémoire primaire).

La taille de la page est souvent déterminée par le (multiple entier du) plus petit multiple entier commun des tailles de blocs considérés, ce afin de minimiser l'espace inutilisable. Cela permettra de répartir aisément un même fichier sur des dispositifs dont la taille des blocs varie sans perte structurelle d'espace.

Dans le cas où tous les dispositifs ont des blocs de même taille, cette tactique demeure pertinente, puisqu'elle permet de réduire de façon sensible le temps d'accès aux données en mémoire secondaire.

Exemples

Calcul de la taille de bloc

Soit trois classes dispositifs ayant respectivement des tailles de blocs de 512, 768 et 1024 octets. La taille de la plage sera un multiple de 6 kio (6144 octets).

Calcul du temps de lecture (moyen)

Soit une taille uniforme de bloc de 1024 octets pour tous les dispositifs, une latence d'accès de 5 ms, un débit de 1024 o/ms. La lecture de 4 kio de données nécessite, en moyenne, 24 ms, si on utilise une page de 1 kio, et 9 ms, si on utilise une page de 4 kio.

Note historique

À une certaine époque, avec des mémoires primaires de l'ordre de 128 kio, on peut comprendre l'intérêt de conserver la taille de pages aussi petite que possible. Il n'était pas rare en effet que la taille de la page soit précisément celle du bloc. Cette époque semble désormais révolue.

2.5. SGBD

SGBR	type organisation
Oracle	fichier non ordonné ou ordonné
DB2	fichier non ordonné
SQL Serveur	fichier non ordonné ou ordonné
PostgreSQL	fichier non ordonné

3. Type d'index

3.1. Index primaire

Un index primaire est défini sur un fichier **ordonné** par un champ **clé**.

L'index est une table ordonnée à deux colonnes *<K(i), P(i)>* :

- K(i) : la clé de l'index correspond à la clé déterminante,
- P(i) : le pointeur vers la page contenant les enregistrements où la valeur du champ clé du premier enregistrement de la page est égale à la valeur de la clé de l'index.

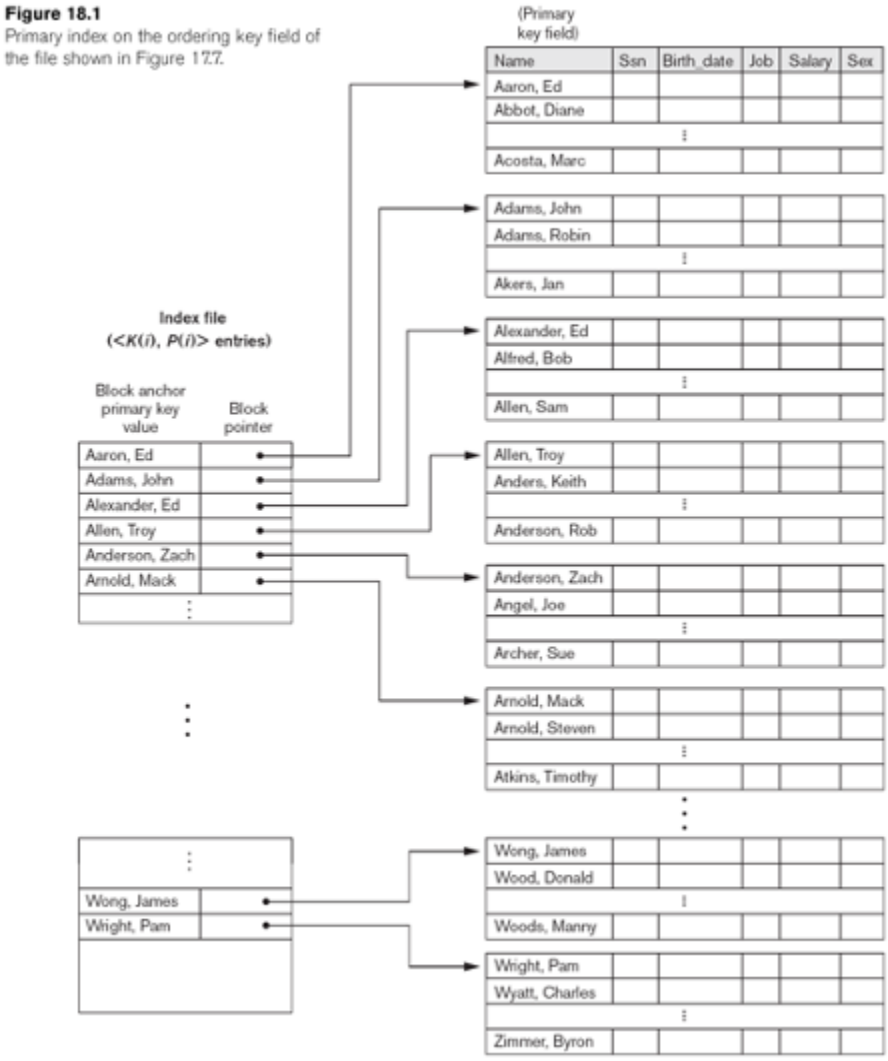

Figure 4. Illustration d'un index clé [Elmasri2016] p?XX.

Exemple
<K(1), P(1)> = <(Aaron Ed), (adresse de la page 1)>
<K(2), P(2)> = <(Adams John), (adresse de la page 2)>

L'index contient autant d'entrées que nombre de valeur du champ clé.

La taille de l'index est plus petite que la taille du fichier de données.

Ce type d'index est créé automatiquement pour la clé primaire de chaque relation.

3.2. Index de classification

Un index de classification (*clustering index*) est défini sur un fichier **ordonné** par un champ **non-clé**.

L'index est une table ordonnée à deux colonnes *<K(i), P(i)>* :

- K(i) : la clé de l'index correspond au champ non-clé,
- P(i) : un pointeur vers la première page contenant les enregistrements ayant la même valeur de la clé de l'index.

L'index contient autant d'entrées que nombre de valeur distincte du champ non-clé. Lorsque la page « déborde », un pointeur vers une page virtuellement contigüe est ajouté.

La taille de l'index est plus petite que la taille du fichier de données.

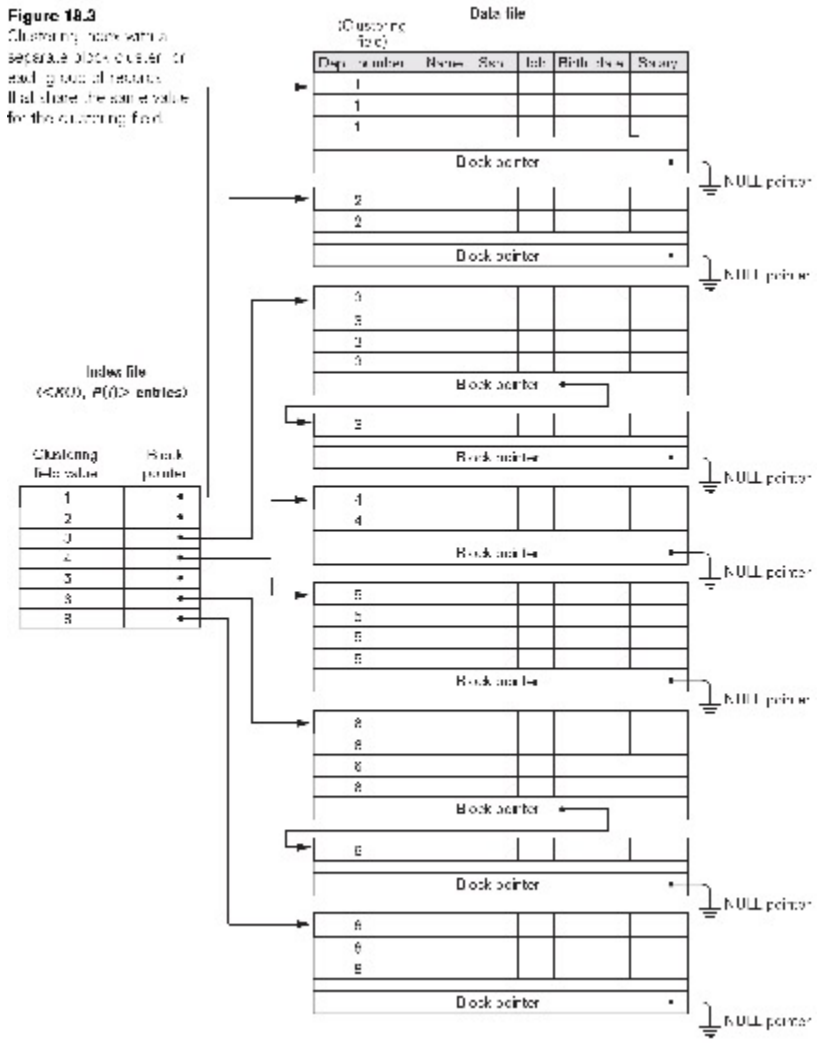

Figure 5. Illustration d'un index de classification [Elmasri2016] p?XX.

3.3. Index secondaire

Un index secondaire est défini sur un fichier ordonné, non-ordonné ou haché.

L'index est une table ordonnée à deux colonnes *<K(i), P(i)>* :

- K(i) : La clé de l'index est un champs clé ou non-clé qui **n'est pas utilisé** pour ordonner les enregistrements du fichier,
- P(i) : un pointeur vers une page de pointeurs d'enregistrements.

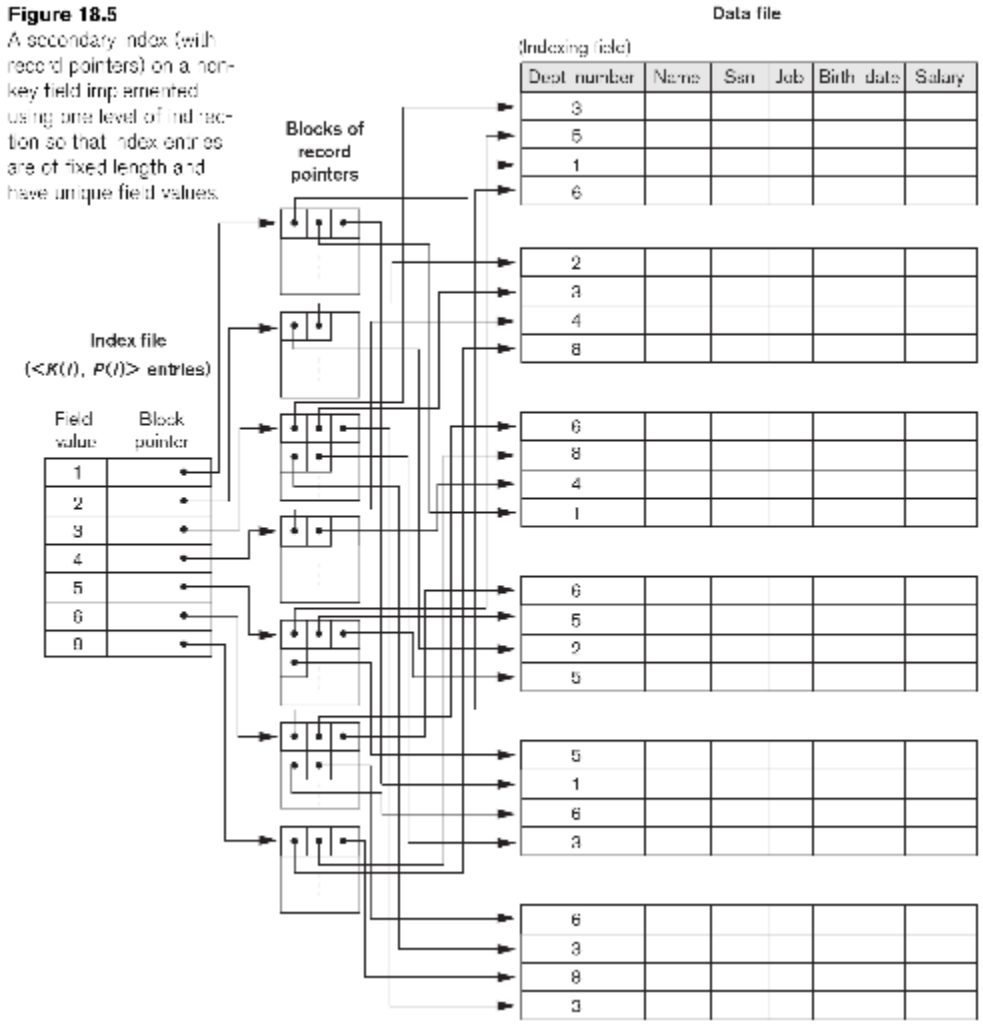

Figure 6. Illustration d'un index secondaire [Elmasri2016] p?XX.

3.4. Index de couverture

Un index de couverture (*covering index*) est défini sur un fichier ordonné ou non ordonné.

L'index est une table ordonnée à trois colonnes *<K(i), A(i), P(i)>* :

- K(i) : la clé de l'index correspond au champ clé,
- A(i) : un ensemble d'attributs associés à la clé,
- P(i) : un pointeur vers la première page contenant les enregistrements ayant la même valeur de la clé de l'index.

Cet index permet de faire la recherche sur la clé et de récupérer un ensemble d'attributs sans devoir lire l'enregistrement au complet.

L'index contient autant d'entrées que nombre de valeur du champ clé.

La taille de l'index est plus petite que la taille du fichier de données.

3.5. Autre type index

- Index pour les intervalles (*GiST — Generalized Search Tree index*)
- Index pour à plusieurs dimensions, comme les données spatiales (R-tree, SP-GIST)
- Index pour la recherche textuelle (*GIN — Generalized Inverted Index*) *…

4. Construction d'un index

4.1. Index sur un attribut

Exemple

```
create index etudiant_nom
  on Etudiant (nom);
```

4.2. Index sur un attribut calculé

Exemple

```
create index etudiant_lower_nom
  on Etudiant (lower(nom));
```

L'utilisation d'une fonction pour calculer une valeur d'un attribut peut empêcher l'utilisation des index.

L'ensemble d'attributs d'un index ne correspond pas nécessairement aux valeurs stockées dans la relation, mais peut être une fonction ou une expression scalaire calculée à partir d'un ou plusieurs attributs de la relation.

4.3. Index sur plusieurs attributs

Exemple

```
create index resultat
  on Resultat (matricule, sigle, session);
```

Un index sur plusieurs attributs peut être construit si les restrictions sont souvent formées de plusieurs attributs.

L'ordre des attributs est important lors de la définition de ce type d'index. Un index sur l'ensemble d'attributs (X, Y, Z) peut être utilisé pour les conditions sur X, XY, XZ ou XYZ, mais pas pour Y ou YZ.

Dans certains cas, cet index offre la possibilité d'effectuer un balayage d'index (sans lire les données sur disque) pour trouver les données demandées.

5. Structure d'indexation

5.1. Tableau de bits

Un index de tableaux de bits (*bitmap index*) est un tableau de données binaires construit par rapport à une valeur de l'attribut indexé.

Ce tableau comporte un élément pour chaque tuplet de la relation indexée. Pour une relation de n tuplets, le tableau est formée de n bits. Le bit i est mis à 1 si le tuplet i a la valeur v pour l'attribut, sinon il est mis à 0.

Un index bitmap est préférable lorsque l'attribut à indexer a une faible cardinalité. Par exemple, les années sont généralement peu nombreuses et ordonnées.

Un index bitmap permet d'effectuer des recherches sur une combinaison de conditions avec les opérateurs suivants :

```
<   <=   =   >=   >
```

Cela inclut les opérateurs SQL : `BETWEEN`

5.2. Arborescence de recherche

Un index d'arbre B (B-Tree index) est une structure de données en arbre équilibré. L'arbre est constitué de nœuds organisés de manière hiérarchique selon la valeur d'un champ.

Un index d'arbre permet une exécution des opérations en temps toujours logarithmique ($n \log n$). En suivant un pointeur, nous limitons notre recherche à chaque niveau à un sous-arbre jusqu'à ce que nous atteignions la page de fichiers de données qui contient les enregistrements requis.

Un arbre de recherche d'ordre p est un arbre tel que chaque nœud contient au maximum p -1 valeurs de recherche et p pointeurs dans l'ordre <P1, K1,..., Pq-1, Kq-1> où q ⇐ p, Pi est un pointeur sur un nœud enfant (ou un pointeur NULL) et Ki est une valeur de recherche parmi un ensemble ordonné de valeurs.

Il existe plusieurs variantes :

- B tree où les pointeurs vers les enregistrements existent dans tous les noeuds
- B+ tree où les pointeurs vers les enregistrements existent seulement dans les feuilles.
- B* tree où les pointeurs vers les enregistrements existent XXX

Un index B-Tree peut être utile sur les attributs de type ordinal pour traiter les conditions avec les opérateurs suivants :

```
<   <=   =   >=   >
```

Cela inclut les opérateurs SQL : `BETWEEN`, `IN`, `IS NULL` , `IS NOT NULL`

5.3. Adressage dispersé

Un index par adressage dispersé est une structure de table à deux colonnes. La première colonne est la clé de hachage et la deuxième colonne est l'adresse vers la page qui contient l'enregistrement.

Un index d'arbre permet une exécution des opérations en temps presque constant.

Un index par adressage dispersé peut être utile sur les attributs de n'importe quel type, mais ne traite que les conditions d'égalité.

Il existe deux techniques d'adressage dispersé (*hashing*) :

- Adressage dispersé statique (*static hashing*) : où le nombre de partitions est fixé à l'avance.
- Adressage dispersé dynamique (*dynamic hashing* ou *extended hashing*) : où le nombre de partitions varie sur les données.

6. Mise en oeuvre

6.1. PostgreSQL

Syntaxe de création d'un index PostgreSQL

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [ IF NOT EXISTS ] name ] ON [ ONLY ]
table_name [ USING method ]
    ( { column_name | ( expression ) } [ COLLATE collation ] [ opclass [ (
opclass_parameter = value [, … ] ) ] ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, …] )
```

```
    [ INCLUDE ( column_name [, …] ) ]
    [ NULLS [ NOT ] DISTINCT ]
    [ WITH ( storage_parameter [= value] [, … ] ) ]
    [ WHERE predicate ]
```

Par défaut, la commande crée un B-Tree.

Il est possible de choisir le type en spécifiant le type après le mot clé `using` (btree, hash, gist, spgist, gin, brin)

Il est conseillé de choisir un nom qui rappelle le but de l'index. Ce nom va apparaitre dans le plan de la requête.

PostgreSQL crée automatiquement un index d'unicité à la déclaration d'une contrainte d'unicité ou d'une clé primaire sur une table.

Créer un index sur une relation de «grande» taille peut prendre beaucoup de temps. Par défaut, PostgreSQL autorise la lecture (SELECT) sur la relation pendant la création d'un index sur celle-ci, mais interdit les écritures (INSERT, UPDATE, DELETE). Elles sont bloquées jusqu'à la fin de la construction de l'index.

Une fois un index créé, le système met à jour l'index lorsque la relation est modifiée et utilise l'index dans les requêtes si cela améliore la performance.

Conclusion

Les facteurs à considérer pour choix des index :

- attributs de jointure (clés référentielles, clés strictes, autres)
- attributs de comparaison (égalité, ordonnancement, intervalles)
- type (consultation, insertion, retrait, modification)
- exigence de performance
- fréquence d'utilisation

À suivre SGBD_03-Execution-et-Optimisation

Références

Compléments au document

- [Elmasri2016a], chapitres 16 et 17
- [Hainaut2022], chapitre 4
- [Lelarge2023a], chapitre 3
- [Petrov2019], chapitre 2, 4 et 6
- [Sciore2020a], chapitre 12
- [Ullman2009a], chapitres 13 et 14

Références

[Dombrovskaya2021a]
Henrietta DOMBROVSKAYA, Boris NOVIKOV et Anna BAILLIEKOVA;
PostgreSQL Query Optimization;
Apress, Berkeley (CA, US), 2021;
ISBN 978-1-4844-6884-1.

[Elmasri2016]
Ramez ELMASRI et Shamkant B. NAVATHE;
Fundamentals of database systems;
7th Edition, Pearson, Hoboken (NJ, US), 2016;
ISBN 978-0-13-397077-7.

[Hainaut2022]
Jean-Luc HAINAUT;
Bases de données - concepts, utilisation et développement;
5e édition, Dunod, 2022;
ISBN 978-2-10-084285-8.

[Lelarge2023a]
Guillaume LELARGE, Juilen ROUOHAUD;
PostgreSQL : architecture et notions avancées;
5e édition, Éditions D_Booker, 2023;
ISBN 978-2-8227-1124-1.

[Petrov2019]
Alex PETROV;
Datavase internals - A deep dive into how distributed data systems work;
1re édition, 3e révision, O'Reilly, 2020;
ISBN 978-1-492-04034-7.

[Sciore2020a]
Edward SCIORE;
Database design and implementation;
Second edition, Springer, 2020;
ISBN 978-3-030-33855-0.

[Ullman2009a]
Hector GARCIA-MOLINA, Jeffrey D. ULLMAN, Jennifer WIDOM;
Database Systems: The Complete Book;
Second Edition, Pearson/Prentice Hall, 2009;
ISBN 9780131873254

Références Wikipedia

- Arbre B, (consulté le 2024-09-14) : https://fr.wikipedia.org/wiki/Arbre_B
- Bitmap, (consulté le 2024-09-14) : https://fr.wikipedia.org/wiki/Tableau_de_bits

Références PostgreSQL

[fr] https://docs.postgresql.fr/16/indexes.html (2024-09-17)
[en] https://www.postgresql.org/docs/16/indexes.htm (2024-09-17) ;

Définitions

Sigles

SGBD (Système de gestion de bases de données)

Service informatique permettant de stocker, manipuler, gérer et partager des données, à l'aide d'un langage fondé sur un modèle permettant d'abstraire la complexité des opérations internes requises tout en garantissant la qualité, la pérennité et la confidentialité des données.

SGBDR (Système de gestion de bases de données relationnelles)

SGBD utilisant un modèle d'abstraction fondée sur la théorie relationnelle de Codd et intégrant comme critère de qualité le maintien des propriétés ACID.

SQL (*Structure Query Language*)

Langage de programmation axiomatique fondé sur un modèle inspiré de la théorie relationnelle proposée par E. F. Codd.
[Normes applicables : ISO 9075:2016, ISO 9075:2023]

Produit le 2025-10-01 10:24:41 UTC

Université de Sherbrooke