Module 3 Revue de schémas de bases de données

IGE 487 Modélisation de bases de données

Chargé de cours : Tudor Antohi

Objectifs de module 3

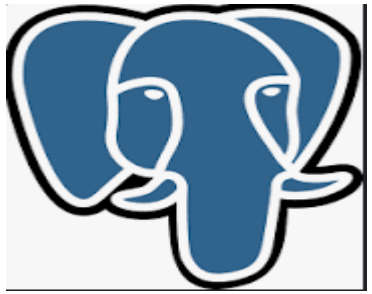
- Revue de la conception et l’exploitation d’une base de données
- Revue de schémas externes et internes
- Schémas internes BD , modèle physique de stockage et indexation
- Schémas interne BD , les objets de catalogue BD de métadonnées
- Schémas interne BD , la sécurité

Conception et exploitation d’une base de données

- Cycle de vie de logiciel BD
- Cycle de vie de modèle logique et conceptuel BD
- Cycle de vie de l’instance physique BD
- Cycle de vie de modèle physique
- Cycle de vie de code applicatif dans la base de données
- Cycle de vie des ensembles de données
- Journalisation des activités BD
- Gestion de la sécurité BD
- Gestion de la performance BD
- Architecture de solution BD
- Gouvernance BD

Cycle de vie logiciel BD

- Installation logiciel BD
- Mise a jour de versions
- Correctifs logiciels
- Correctifs de sécurité
- Ajout des extensions
- Rollbacks de versions ou correctifs



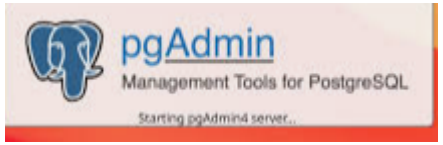
ORACLE

Cycle de vie de l'instance physique

Une instance BD comprend l'environnement SGBD et tous les composants: système d'opération, gestionnaire de mémoire physique et gestionnaire de stockage et le catalogue des objets.

- Création de l'instance ; une instance BD a plusieurs bases de données
- Création de bases de données physique
- Backup et recouverte d'instance ou base de données
- Mise a jour de la configuration instance et base de données
- Destruction d'instance et de la base de données
- Déploiement entre les environnements

Divers outils d'administration supporte les activités.



Cycle de vie de modèle logique et conceptuel BD

- Modelé conceptuel (ex.: modèle info-relationnel a haut niveau)
- Modelé logique détaillé (ex.: définition des tables utilisant les techniques de normalisation 1FN-6FN)
- Représentation des architectures de données utilisant UML et divers outils de modélisation.

Visio 2021 Professional

IDERA – ER/Studio Data Architect

erwin by Quest

ENTERPRISE ARCHITECT SPARX SYSTEMS

PowerAMC

Modélisation logique d’une BD

Le modèle conceptuel contient le schéma conceptuel de base de données, spécifié lors de la conception de la base de données. Il est orienté sur les besoins d’affaires et il est Independent de la technologies utilisée. Même s’il est un modèle a haut nveau il doit décrire complètement la sémantique de modèle d’affaires.

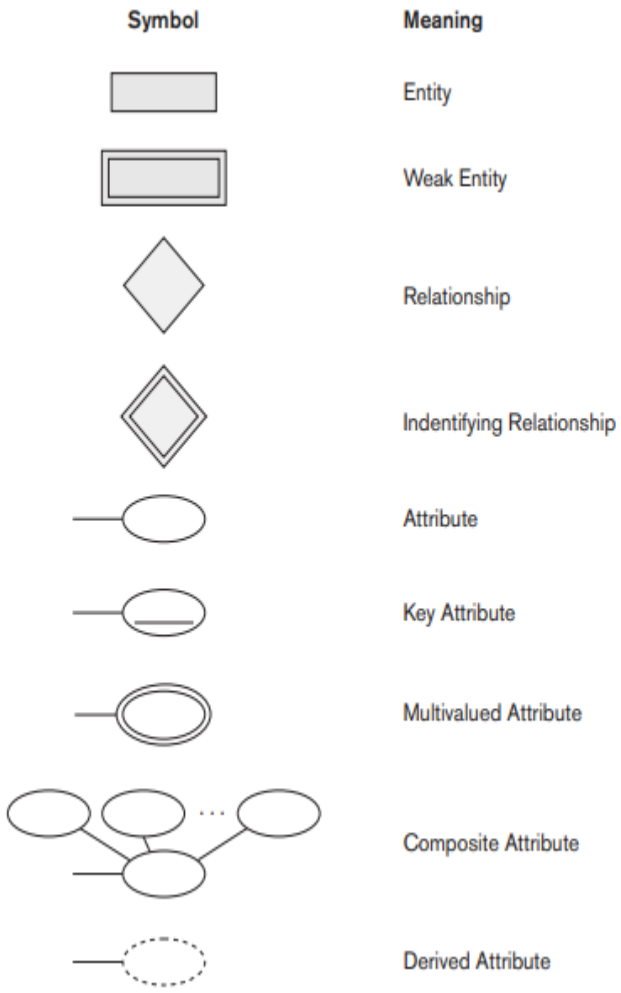
Le schéma conceptuel masque les détails des structures de stockage physiques et se concentre sur la description des entités, des types de données, des relations, des opérations de l'utilisateur et des contraintes.

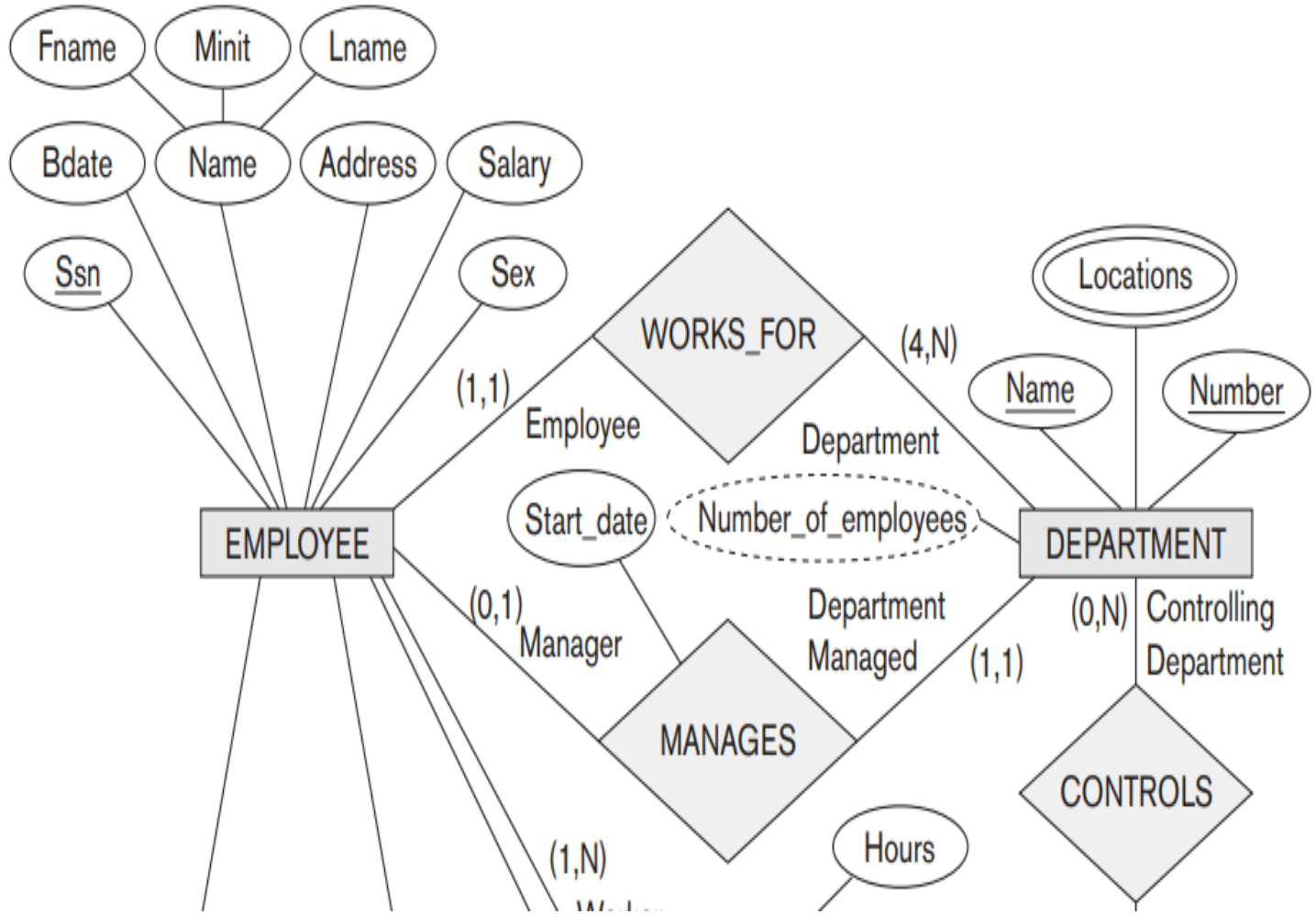
La plupart des modèles de données ont certaines conventions pour afficher les schémas sous forme de diagrammes.

Blocs utilisés dans les modèles logiques

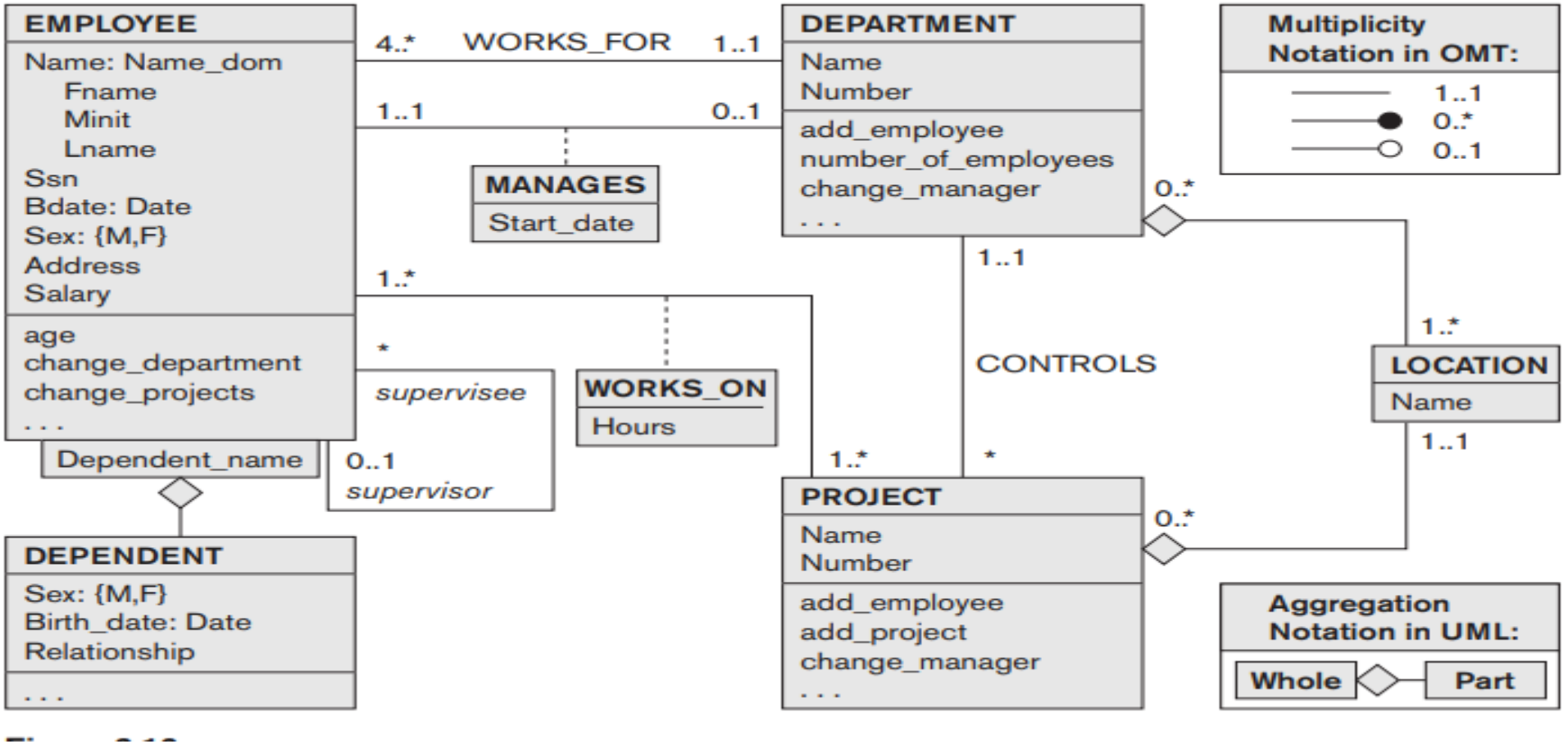
Diagrammes Entité Relation	Bloc physique	Bloc XML	Bloc orienté objet (UML)
Entité	Table	Élément	Classe
Attribut	Colonne	Attribut	Membre (excluant les méthodes)
Ensemble de relations	Clé référentielle	XREF	Association, agrégation, composition
Identificateur	Clé primaire	Location en XML	GUID

Diagrammes Entité – Relation

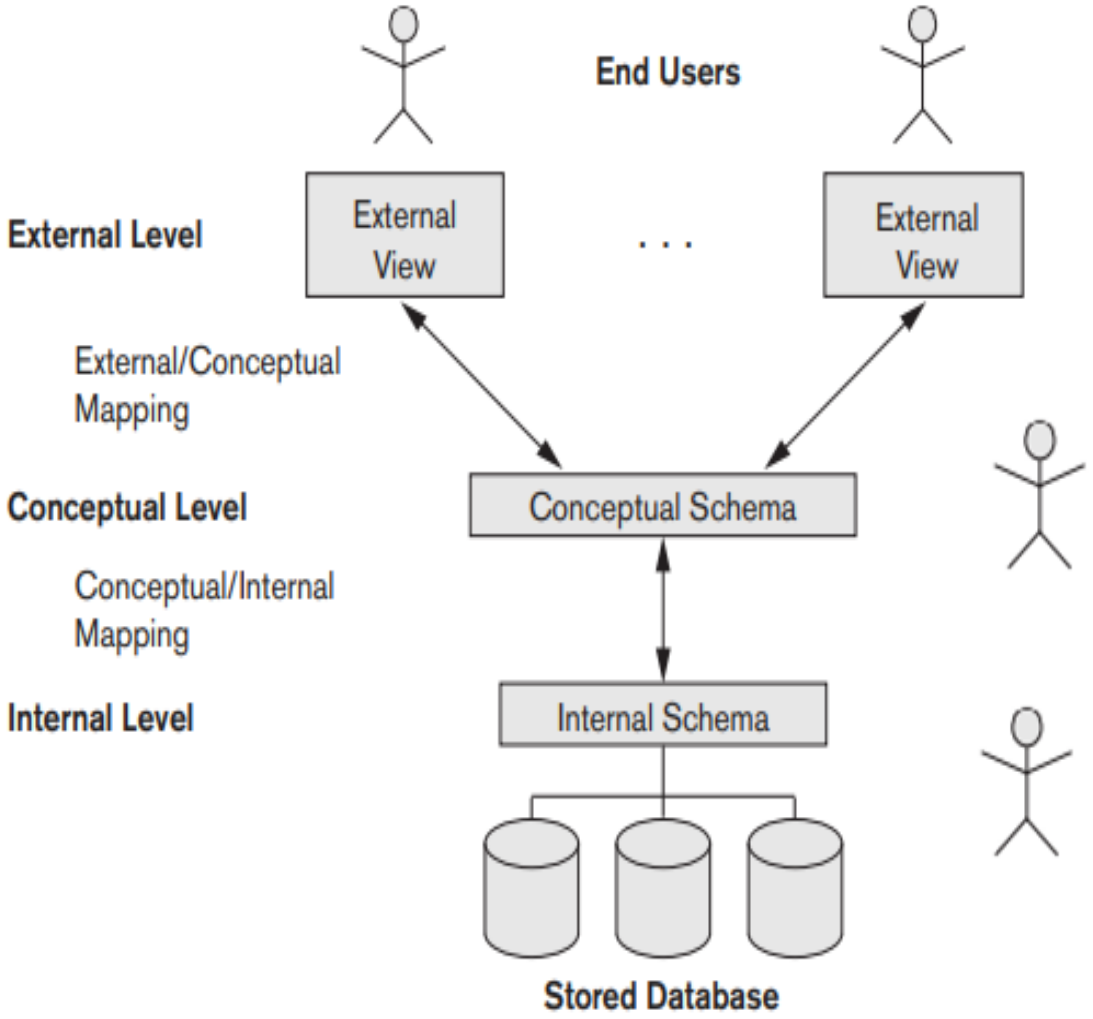




Diagrammes UML



Les niveaux externes, conceptuelles et internes



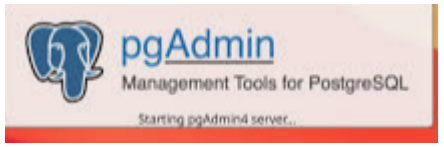
Architecte de données

Administrateur de base de données

Cycle de vie de modèle physique

Le modèle physique (schéma interne), décrit la structure de stockage physique de la base de données, les détails complets du stockage des données et les chemins d'accès vers les données de la base de données.

- Création des schémas, objets (tables, vues, indexes, tablespaces, fichiers)
- Mise a jour des objets et leurs propriétés
- Destruction des objets
- Déploiement entre les environnements



Cycle de vie de code applicatif BD

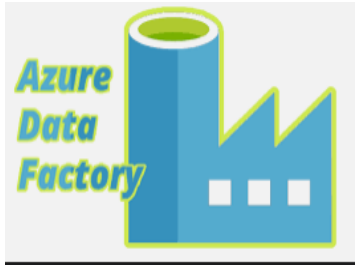
Les bases de données contiennent du code applicatif : triggers, scripts, procédures stockées et fonctions. Le code est représenté comme des objets dans le catalogue de la base de données.

- Création et destruction du code
- Mise a jour
- Déploiement
- Teste

Cycle de vie des ensembles de données

Les ensembles de données BD sont crées, transportées et traitées entre les environnements applicatifs par les développeurs ETL et les scientifiques de données.

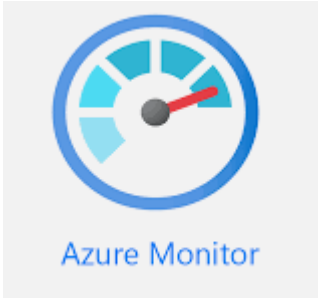
- Chargement de données
- Exports de données
- Traitements de qualité de données



Journalisation des activités BD

Les activités et les évènements BD sont surveillées pour les analyses opérationnelles et diagnostiques.

- Arrêts et démarrages BD et instances
- Croissance d'espace
- Alertes d'environnement BD (système d'opération, stockages externes)
- Audit de DDLs et DCLs



Performance BD

- **Surveillance de performance** – évolution vers l'intelligence artificielle, analyses descriptives, diagnostique, recommandation et prédictions.
- **Optimisation de la performance**
  - Optimisation de fonctionnement d'instance BD (ex. ajout de mémoire, mise a jour de stockage, paramètres)
  - Optimisation du modèle physique
  - Optimisation de requetés SQL (prochaine cours)
  - Optimisation du code applicatif (ex.: gérer mieux les transactions)

Architecture de solutions BD

- Feuilles de route
- Gestion de la pérennité
- Choix technologiques
- Solutions de haute disponibilité
- Solutions de réplication
- Solutions d'intégration dans le contexte d'entreprise

Gartner

Obs.: a reprendre certaines notions dans le module de systèmes distribués.

Gouvernance BD

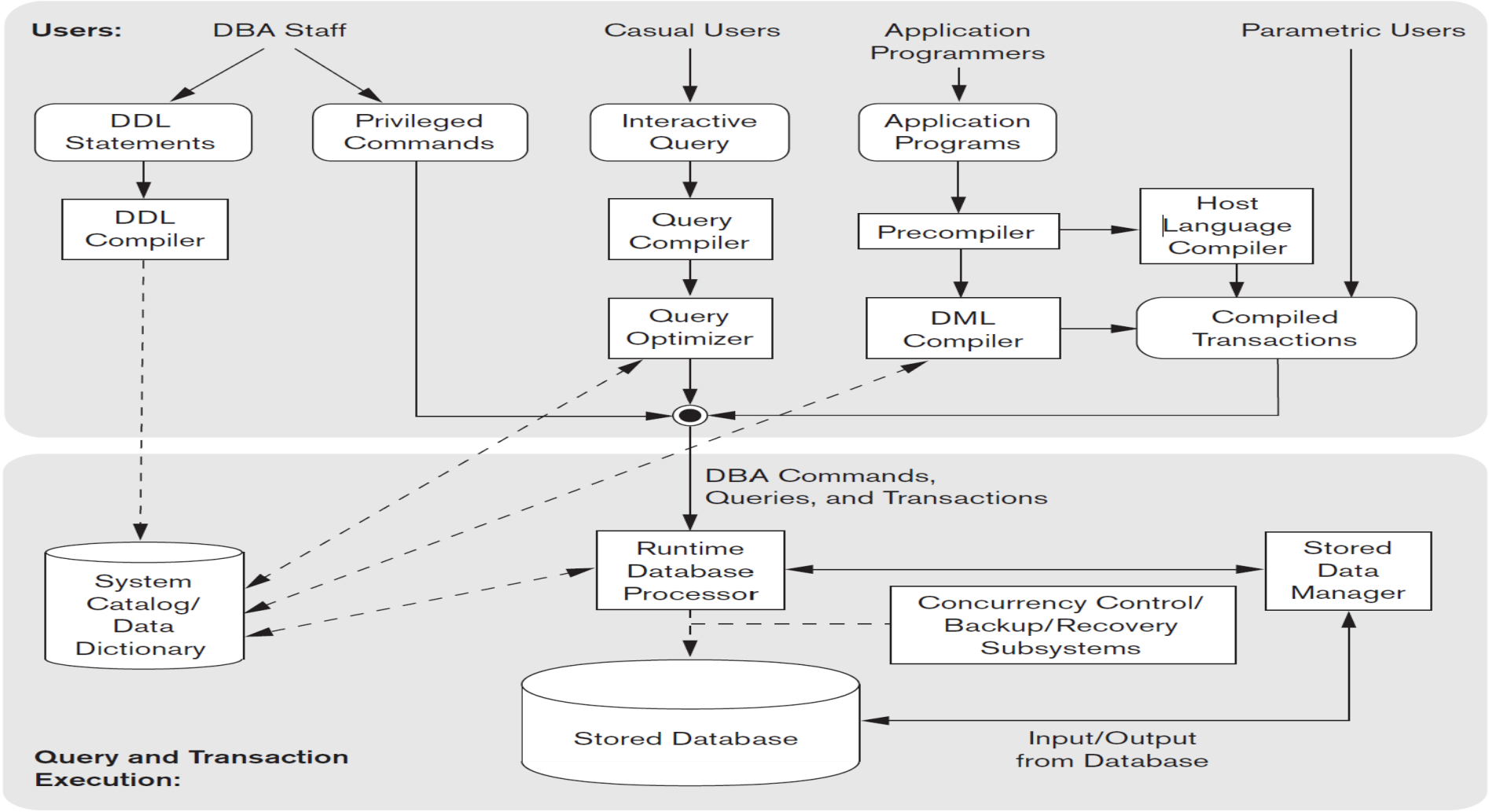
- Gère les assujettissements de données et bases de données selon les normes internes de la compagnie et les lois en vigueur.
- Gestion des couts
- Concepts de data marketing

Ex.: Loi 25, assujettissements NERC, RGPD, Privacy by Design

Sécurité BD

- Gestion des accès
- Gestion des privilèges
- Audit

Base de données



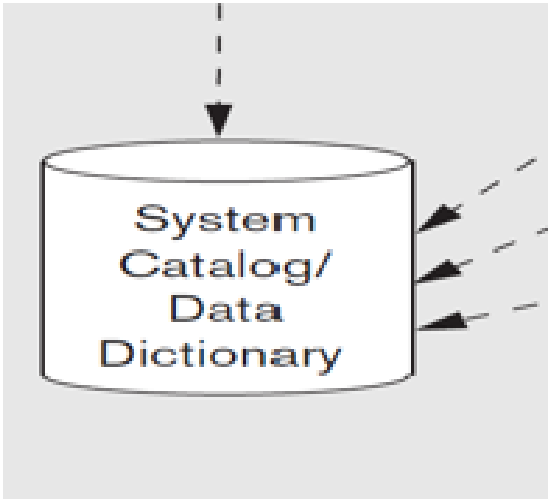
Catalogue des données

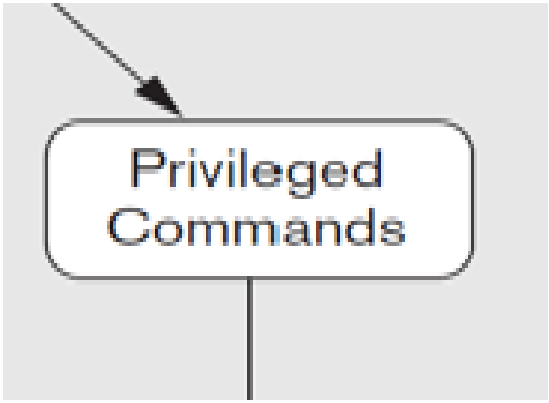
- Le compilateur DDL traite les définitions de schéma, spécifiées dans le DDL, et stocke les descriptions des schémas (métadonnées) dans le catalogue du SGBD.

- Le catalogue comprend des informations (métadonnées) sur :
  - tous les objets de schémas et leurs propriétés
  - noms et les tailles des fichiers, détails de stockage
  - détails de sécurité (privilèges, comptes, accès)

Catalogue des données et Sécurité

- **Objets BD – langage DDL**
  - Data Types
  - Tables , Vues
  - Clés, Contraintes
  - Indexes
  - Autres objets …
- **Sécurité BD – langage DCL**
  - Cycle de vie de comptes et accès
  - Cycle de vie de privilèges
  - Polices , bases de données virtuelles
  - Audit





PostgreSQL – Histoire

PostgreSQL a été développé par l'Université de Californie à Berkeley. La première version a été créé par Michael Stonebraker (prix Turing 2015) et publié en 1994. Deux étudiants, Andrew Yu et Jolly Chen, ont ensuite apporté un sous-ensemble de SQL à Postgres et ont renommé le système en Postgres95. Le système a ensuite été maintenu et développé dans le monde open source en dehors de Berkeley, et finalement renommé PostgreSQL.

PostgreSQL est gratuit, sous licence open-source. Il est maintenu en permanence depuis 1996 par le PostgreSQL Global Development Group.

https://www.postgresql.org/community/contributors/

PostgreSQL – Modèle de données

PostgreSQL est une base de données relationnelle objet (une base de données relationnelle avec un modèle de base de données orienté objet). Les objets sont gères dans les schémas de base de données.

La base de données relationnelle objet hérite la mathématique relationnelle avec la possibilité de définir des types de données complexes.

Nouveaux types de tous les objets à l'intérieur de PostgreSQL peuvent être créés, y compris : la conversion, la distribution, les fonctions, les types de données, les domaines, les procédures et les index.

PostgreSQL – Stockage

PostgreSQL stocke la table et son index sur le disque.

La mémoire volatile (buffer) est utilisée pour accélérer les requêtes. La taille par défaut est de 128 mégaoctets, qui peut être modifiée dans la configuration d'exécution. L'utilisateur peut également modifier la mémoire dédiée aux opérations internes telles que le tri, au cas où les 4 mégaoctets par défaut seraient insuffisants.

Chaque table et son index sont stockés séparément dans des fichiers. Le fichier est nommé par le numéro de nœud de fichier de sa table ou de son index défini dans le catalogue. Une mappe d'espace libre est conservée pour stocker des informations sur l'espace disponible. Le fichier est divisé en segments séparés une fois que sa taille depasse 1 Go.

PostgreSQL – catalogue

- Les catalogues sont l'endroit où un SGBD relationnel stocke les métadonnées de schéma, telles que les informations sur les tables et les colonnes, et les informations de configurations internes.
- Les catalogues système de PostgreSQL sont des tables régulières. Il est possible de supprimer et recréer les tables, ajouter des colonnes, insérer et mettre à jour des valeurs et perturber gravement votre système de cette façon.
- Il ne faut pas modifier les catalogues système à la main, il existe des commandes SQL pour le faire. (ex.: CREATE DATABASE insère une ligne dans le catalogue **pg_database**)

PostgreSQL –exemples tables et vues du catalogue

- **Pg_tables**
- **Pg_indexes**
- **Pg_tablespaces**
- **Pg_config** - La vue pg_config décrit les paramètres de configuration au moment de la compilation de la version actuellement installée de PostgreSQL.
- **Pg_type**
- **Pg_user**
- **Pg_views**

Stockages (1)

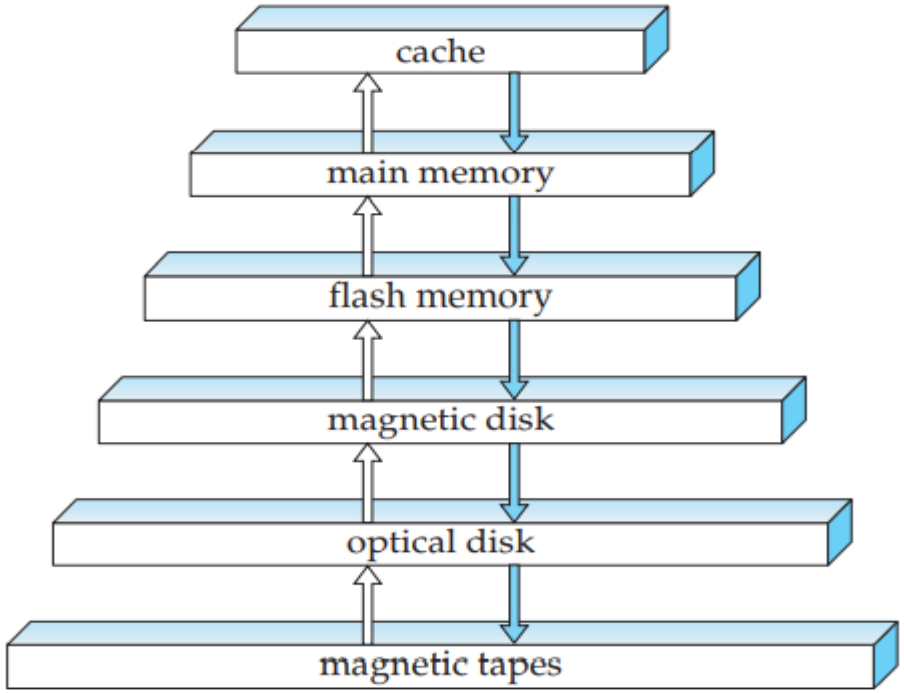
- **Cache :** la plus rapide forme de stockage. Les implémentations de base de données tiennent compte de cette particularité pour l'optimisation de la performance.
- **Mémoire principale :** Les instructions machine fonctionnent dans la mémoire principale. Bien que la mémoire principale puisse contenir plusieurs giga-octets de données sur un ordinateur personnel, voir des centaines de giga-octets de données dans de grands systèmes de serveurs, elle est généralement trop petite (ou trop chère) pour stocker la totalité de la base de données.

Stockages (2)

- **Flash :** Il existe deux types de mémoire flash, NAND et NOR. Le flash NAND a une capacité de stockage plus élevée est largement utilisé pour le stockage de données dans des appareils tels que des appareils photo, des lecteurs de musique et des téléphones portables, et de plus en plus également dans des ordinateurs portables. La mémoire flash est utilisée en remplacement des disques magnétiques
- **Disques magnétiques :** Le support principal pour le stockage en ligne des données. La base de données est stockée sur un disque magnétique. Le système déplace les données du disque vers la mémoire principale afin qu'ils soient accessibles. Une fois que le système a effectué les opérations désignées, les données qui ont été modifiées doivent être écrites sur le disque

Stockages (3)

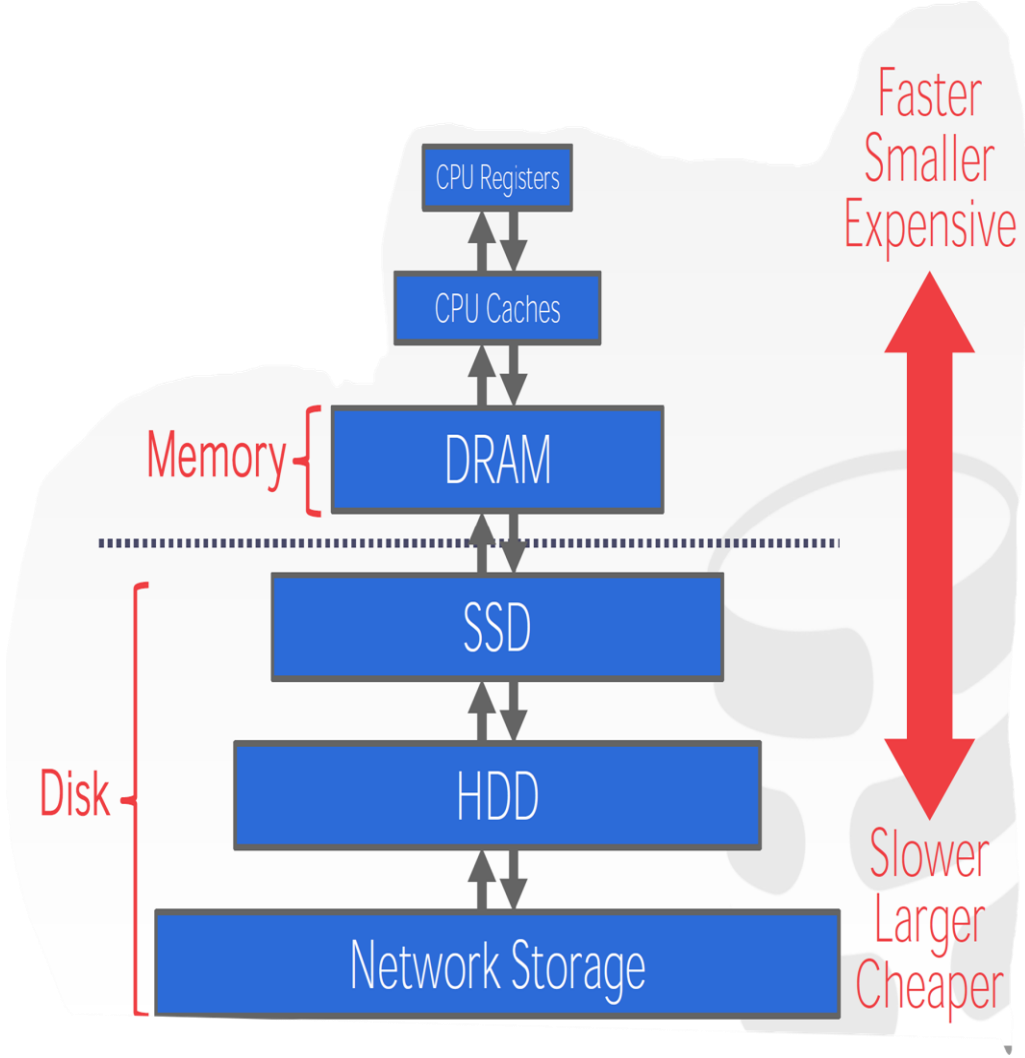
- **Bande magnétique :** principalement utilisé pour la sauvegarde et l'archivage des données. La bande magnétique est moins chère mais l'accès séquentiel aux données est plus lent.



Stockages modernes

- **SAN –** réseaux de stockage , basé sur la communication optique , les périphériques de stockage sont configurés comme des nœuds sur un réseau à haut débit et peuvent être attachés et détachés des serveurs de manière flexible.
- **NAS –** partage de fichiers en réseau par de serveurs de fichiers, qui ne fournissent aucun des services de serveur, mais permettent simplement l'ajout de stockage pour le partage de fichiers.
- **iSCSI –** ne nécessite pas le câblage Fibre Channel et peut fonctionner sur de longues distances en utilisant l'infrastructure réseau existante. En transportant les commandes SCSI sur les réseaux IP, iSCSI facilite les transferts de données sur les intranets et gère le stockage sur de longues distances. Il peut transférer des données sur des réseaux locaux (LAN), des réseaux étendus (WAN) ou Internet.

Stockages – temps d’accès , prix , durée de vie



ACCESS TIMES

0.5 ns	L1 Cache Ref	0.5 sec
7 ns	L2 Cache Ref	7 sec
100 ns	DRAM	100 sec
150,000 ns	SSD	1.7 days
10,000,000 ns	HDD	16.5 weeks
~30,000,000 ns	Network Storage	11.4 months
1,000,000,000 ns	Tape Archives	31.7 years

Stockages infonuagique Azure

- **Azure fournit 4 types de stockage persistent**
  - Ultra SSD (64TB, 160K IOPS per disque, débit 2000MB/s)
  - Premium SSD
  - Standard SSD
  - Standard HDD (disques magnétiques)

Ultra SSD Managed Disk Offerings

Disk size (GiB)	4	8	16	32	64	128	256	512	1,024-65,536 (in increments of 1 TiB)
IOPS range	100-1,200	100-2,400	100-4,800	100-9,600	100-19,200	100-38,400	100-76,800	100-153,600	100-160,000
Throughput Cap (MBps)	300	600	1,200	2,000	2,000	2,000	2,000	2,000	2,000

Hiérarchisation automatisée de stockage

- L'administrateur de stockage peut configurer une politique de hiérarchisation dans laquelle les données moins fréquemment utilisées sont déplacées vers des disques plus lents et moins chers et les données les plus fréquemment utilisées sont déplacées vers des disques plus rapides comme SSD.
- Un principe similaire s’applique pour les stockage chaude (SSD), tiède (disques magnétiques) et froids (bande, autres stockages comme AWS Glacier)

Stockage objet

Les systèmes basés sur des fichiers orientés matériel évoluent vers de nouvelles architectures. Le dernier en date est le stockage basé sur des objets.

- Dans ce schéma, les données sont gérées sous la forme d'objets plutôt que de fichiers constitués de blocs. Les objets transportent des métadonnées qui contiennent des propriétés qui peuvent être utilisées pour gérer ces objets. Chaque objet porte un identifiant global unique qui est utilisé pour le localiser.
- le stockage d'objets force le verrouillage au niveau de l'objet, il n'est pas bon pour les traitement des transactions à haut débit. Par conséquent, il n'est pas viable pour les applications de base de données de niveau entreprise.

Ex.: AWS S3, Azure BLOB, excellent choix pour conservation de données, et données en lecture seulement.

Stockages infonuagique Azure et Amazon – 5mins

- **Présentation sur Azure Pricing Calculator**

https://azure.microsoft.com/en-us/pricing/calculator/

- **Présentation sur AWS Pricing Calculator**

https://calculator.aws/#/

Fichiers

- Les données stockées sur disque sont organisées sous forme de fichiers d'enregistrements.
- Chaque enregistrement est une collection des faits sur les entités, leurs attributs et leurs relations.
- Les enregistrements sont stockés sur disque de manière à pouvoir les localiser efficacement lorsqu'ils sont nécessaires.

Organisations de fichiers

Il existe plusieurs **organisations primaires** de fichiers, qui déterminent comment les enregistrements de fichiers sont physiquement placés sur le disque, et accessibles.

- **heap** (non ordonné) place les enregistrements sur le disque sans ordre particulier en ajoutant de nouveaux enregistrements à la fin du fichier,
- **trié (ou fichier séquentiel)** conserve les enregistrements classés par la valeur d'un champ particulier (appelé la clé de tri).
- **hash** utilise une fonction de hachage appliquée à un champ particulier (appelée clé de hachage) pour déterminer l'emplacement d'un enregistrement sur le disque.
- **arbres B**, utilisent des structures arborescentes.

Une **organisation secondaire** permet un accès efficace aux enregistrements de fichiers basés sur des champs alternatifs à ceux qui ont été utilisés pour l'organisation de fichier principal. La plupart d'entre eux sont des ***indexes.***

Fichiers Heap (1)

Les enregistrements sont placés dans l'ordre dans lequel ils sont insérés, les nouveaux enregistrements sont insérés à la fin du fichier.

Inserts - *rapides a la fin de fichier*

Recherches - *lentes, séquentielles*

Suppression - *un programme doit chercher le bloc, copier le bloc dans la mémoire, supprimer l'enregistrement et réécrire le bloc sur le disque.*

La suppression d'un grand nombre d'enregistrements entraîne un gaspillage d'espace de stockage.

Une autre technique consiste à stocker un octet ou un bit supplémentaire, un marqueur de suppression et le changer au besoin.

Fichiers Heap (2)

Il peut exister une organisation fractionnée ou non fractionnée pour un fichier non ordonné, avec des enregistrements (blocs logiques) de longueur fixe ou variable.

Fractionnement : *les blocs logiques ne sont pas égales avec les blocs physiques*

Danger de la longueur variable : *La modification d'un enregistrement de longueur variable peut nécessiter la suppression de l'ancien enregistrement et l'insertion d'un enregistrement modifié car l'enregistrement modifié n’entre pas dans son ancien espace sur le disque.*

Fichiers Heap (3)

Pour un fichier d'enregistrements de longueur fixe utilisant des blocs non fractionnés et une allocation contiguë, il est simple d'accéder à n'importe quel enregistrement par sa position dans le fichier.

Si les enregistrements logiques du fichier sont numérotés **0, 1, 2, … , r – 1** et les enregistrements de chaque bloc physique sont numérotés **0, 1, …, bfr – 1, où bfr** est le facteur de blocage, alors le **i** -ème enregistrement est situé dans le bloc (**i/bfr**) et est le (**i mod bfr**) -ème enregistrement de ce bloc.

Un tel fichier est appelé fichier relatif ou direct.

Obs.: Accéder directement à un enregistrement par sa position n'aide pas à localiser un enregistrement en fonction d'une condition de recherche ; cependant, cela facilite la construction de chemins d'accès sur le fichier (utilisation des indexes)

Fichiers tri (1)

Nous pouvons trier physiquement les enregistrements d'un fichier sur disque en fonction des valeurs de l'un de leurs champs, appelé champ de tri. Cela conduit à un fichier ordonné ou séquentiel.

Si le champ de tri est également un champ clé du fichier (un champ garanti pour avoir une valeur unique dans chaque enregistrement), alors le champ est appelé la clé de tri du fichier.

Type of Organization	Access/Search Method	Average Blocks to Access a Specific Record
Heap (unordered)	Sequential scan (linear search)	$b/2$
Ordered	Sequential scan	$b/2$
Ordered	Binary search	$\log_2 b$

Fichiers tri (2)

L'insertion et la suppression d'enregistrements sont des opérations coûteuses pour un fichier ordonné car les enregistrements doivent rester physiquement ordonnés.

- Pour **insérer** un enregistrement, nous devons trouver sa position dans le fichier, en fonction de sa valeur de champ de tri, puis faire de la place dans le fichier pour insérer l'enregistrement à cette position. Pour un fichier volumineux, cela peut prendre beaucoup de temps car, en moyenne, la moitié des enregistrements du fichier doivent être déplacés pour faire de la place pour le nouvel enregistrement.
- Pour la **suppression** d'enregistrements, le problème est moins grave si on utilise des marqueurs de suppression et une réorganisation périodique.

Obs.: *Oracle a une implantation de tables organisés en index.*

Fichiers Hash (1)

Le hachage permet un **accès rapide aux enregistrements** sous certaines conditions de recherche.

La condition de recherche doit être **une condition d'égalité** sur un seul champ, appelé champ de hachage. Dans la plupart des cas, le champ de hachage est également un **champ clé du fichier**, auquel cas il est appelé **clé de hachage**.

L'idée derrière le hachage est de fournir une f**onction h**, appelée fonction aléatoire de hachage qui est **appliquée à la valeur du champ de hachage** d'un enregistrement et **donne l'adresse du bloc de disque dans lequel l'enregistrement est stocké**. Pour la plupart des enregistrements, nous n'avons besoin que d'un accès à un seul bloc pour récupérer cet enregistrement.

Obs.: la valeur logique et convertie dans une valeur physique

Fichiers Hash (2)

Le but d’une fonction de hachage est :

1. **Répartir uniformément** les enregistrements sur l'espace d'adressage afin de minimiser les collisions, permettant ainsi de localiser un enregistrement avec la clé donnée dans un seul accès.
2. Atteindre l’objectif ci-dessus **tout en occupant entièrement les places**, ne laissant pas beaucoup d’espace inutilisé.

- **Hachage interne :** les enregistrements a l’intérieur d’un fichier, **Hachage externe :** les fichiers sur le disque
- **Fonctions d’hachage : dynamique ou linéaire** pour optimiser le overflow, équilibrer la répartition de stockage, les requêtes et les actualisations BD.

Indexes

Structures d'accès auxiliaires (chemins d'accès secondaires) utilisées pour accélérer la recherche des enregistrements en réponse à certaines conditions de recherche, sans affecter le placement physique des enregistrements dans le fichier de données principal sur le disque.

Les indexes utilisent des algorithmes ISAM, B-tree, B*-tree, hachage ou bitmaps.

Un **index dense** a une entrée d'index pour chaque valeur de clé de recherche (et donc chaque enregistrement) dans le fichier de données.

Un **index sparse** (ou non dense), n'a d'entrées d'index que pour certaines valeurs de recherche.

Indexes (2)

- **Indexes primaires** - l'index primaire est spécifié sur le champ clé d'ordre d'un fichier ordonné d'enregistrements. Chaque enregistrement a une valeur unique pour ce champ.
- **Indexes type clusters -** trient et stockent les lignes de données dans la table en fonction de leurs valeurs de clé.
- **Indexes secondaires** - peut être spécifié sur n'importe quel champ non ordonné d'un fichier

Indexes primaires

Un index primaire est un **fichier ordonné** dont les enregistrements sont de longueur fixe avec **deux champs.** Le **premier champ** est du même type de données que le **champ de clé de tri**, appelé clé primaire, du fichier de données. Chaque enregistrement a une valeur unique pour ce champ.
Le **deuxième champ** est un **pointeur vers un bloc** de disque (une adresse de bloc).

<K(1) = (Aaron, Ed), P(1) = address of block 1>
<K(2) = (Adams, John), P(2) = address of block 2>
<K(3) = (Alexander, Ed), P(3) = address of block 3>

Indexes primaire - exemple

Sans index: Supposons u**n fichier ordonné** avec r = 300 000 enregistrements stockés sur un disque avec une taille de bloc B = 4 096 octets. Les enregistrements de fichier sont de taille fixe et non étendus, avec une longueur d'enregistrement R = 100 octets. Le facteur de blocage pour le fichier serait bfr = (B/R)= (4 096/100)= **40 enregistrements** par bloc. Le nombre de blocs nécessaires pour le fichier est b = (r/bfr)= (300 000/40) = 7 500 blocs. Une recherche binaire sur les données le fichier aurait besoin d'environ $\log_2$ b= ($\log_2$ 7 500) = **13 accès au blocs.**

Avec index: Supposons maintenant que le champ de clé d'ordre du fichier ait une longueur de V = 5 octets, qu'un pointeur de bloc ait une longueur de P = 6 octets et que nous ayons construit un index primaire pour le fichier. La taille de chaque entrée d'index est Ri = (5 + 6) = 11 octets, donc le facteur de blocage pour l'index est bfr_i = (B/Ri)= (4 096/11)= **372 entrées par bloc**. Le nombre total d'entrées d'index ri est égal au nombre de blocs dans le fichier de données, qui est de 7 500. Le nombre de blocs d'index est donc b_i = (ri/bfri)= (7 500/273)= 20 blocs. Pour effectuer une recherche binaire sur le fichier d'index, il faudrait ($\log_2$ bi) =($\log_2$ 20) = 4…5 accès au bloc. Pour rechercher un enregistrement à l'aide de l'index, nous avons besoin d'un bloc d'accès supplémentaire au fichier de données pour un total de **5 + 1 = 6 accès aux blocs**

Indexes type cluster (Elmasri)

Au maximum, un index par table peut être un index primaire ou de clustering, car **cela implique que le fichier soit physiquement ordonné sur cet attribut index**. Dans la plupart des SGBDR, cela est spécifié par le mot-clé CLUSTER. (Si l'attribut est une clé, un index primaire est créé, tandis qu'un index de clustering est créé si l'attribut n'est pas une clé.)

- Si une table nécessite plusieurs index, la décision concernant celui qui doit être l'index primaire ou de clustering dépend de si le maintien de la table ordonnée sur cet attribut est nécessaire.
- Les requêtes bénéficient beaucoup du clustering. Si une requête utilise uniquement un index , clustering est une solution. Un index de clustering peut être configuré en tant qu'index multiattribut si le cas.

Indexes type cluster - recapitulation

Si les enregistrements de fichier sont physiquement ordonnés sur un champ non-clé (qui n'a pas de valeur distinctes pour chaque enregistrement), ce champ est appelé le champ de clustering et le fichier de données est appelé un fichier clustérisé.

Exemple

Supposons que nous considérons le même fichier ordonné avec r = 300 000 enregistrements stockés sur un disque avec une taille de bloc B = 4 096 octets. Imaginez qu'il soit classé par l'attribut Code postal et qu'il y ait 1 000 codes postaux dans le fichier (avec une moyenne de 300 enregistrements par code postal, en supposant une répartition uniforme entre les codes postaux.)

L'index a 1 000 entrées d'index de 11 octets chacune (code postal de 5 octets et pointeur de bloc de 6 octets) avec un facteur de blocage bfri = (B/Ri) = (4 096/11) = 372 entrées d'index par bloc. Le nombre de blocs d'index est donc bi = (ri/bfri) = (1 000/372)= 3 blocs.

Pour effectuer une recherche binaire sur le fichier d'index, il faudrait ($\log_2$ bi) = ($\log_2$ 3) = 2 accès au bloc. Cet index serait généralement chargé dans la mémoire principale (11 Ko) et prend un temps négligeable pour effectuer une recherche en mémoire. Un bloc d'accès au fichier de données conduirait au premier enregistrement avec un code postal donné.

Indexes type cluster – commentaire Elmasri

Clé primaire = concept logique, pas necessairement, pas necessairement ordre physique.
Clustered Indez = Table ordonnée physiquement par un B-Tree
En certaines documentations = tous les tables ont un index cluster = FAUX (s'applique a des certines technologies BD)
Oracle utilize le concept de Index Organized Table = Table IOT, on peut avoir des indexes secondaires sur une IOT. En Oracle il existe le concept de cluster qui est different de table ou index cluster, attentiona a la confusion. S**AP-HANA** aussi.
SQL Server et MYSQL – pas des indexes secondaire sur le clustered index.
SQL Server – le concept de columnstore index, la table est ordonnées sur tous les attributs

Indexes secondaires

Un index secondaire fournit un moyen secondaire d'accéder à un fichier de données pour lequel un accès principal existe déjà. Les enregistrements du fichier de données peuvent être ordonnés, non ordonnés ou hachés. L'index secondaire peut être créé sur un champ qui est une clé candidate et a une valeur unique dans chaque enregistrement, ou sur un champ non clé avec des valeurs en double. L'index est un fichier ordonné à deux champs, comme les autres.
Soit **<K(i), P(i)>.** Les entrées sont classées **par valeur de K(i),** nous pouvons donc effectuer une recherche binaire. Étant donné que **les enregistrements du fichier de données ne sont pas physiquement ordonnés** par les valeurs du champ clé secondaire, nous ne pouvons pas utiliser d'ancres de bloc. **Une entrée d'index secondaire est créée pour chaque enregistrement du fichier de données**, plutôt que pour chaque bloc, comme dans le cas d'un index primaire.
Un index secondaire nécessite généralement plus d'espace de stockage et un temps de recherche plus long en raison de son plus grand nombre d'entrées. Cependant, l'amélioration du temps de recherche d'un enregistrement arbitraire est beaucoup plus importante pour un index secondaire que pour un index primaire, puisqu'il faudrait faire une recherche linéaire sur le fichier de données si l'index secondaire n'existait pas. Pour un index primaire, nous pourrions toujours utiliser une recherche binaire sur le fichier principal, même si l'index n'existait pas.

Indexes secondaires - exemple

Considérons le fichier de l'exemple 1 avec r = 300 000 enregistrements de longueur fixe de taille R = 100 octets stockés sur un disque avec une taille de bloc B = 4 096 octets. Le fichier contient b=7 500 blocs, comme calculé dans l'exemple 1. Supposons que nous voulions rechercher un enregistrement avec une valeur spécifique pour la clé secondaire, un champ clé non ordonné du fichier qui fait V = 9 octets de long. Sans l'index secondaire, faire une recherche linéaire sur le fichier nécessiterait b/2 = 7 500/2 = **3 750 accès bloc en moyenne (accès séquentiel)**

Supposons que nous construisions un index secondaire sur ce champ clé non ordonné du fichier. Un pointeur de bloc a une longueur de P = 6 octets, donc chaque entrée d'index est Ri = (9 + 6) = 15 octets, et le facteur de blocage pour l'index est bfri = (B/Ri) = (4 096/15) = **273 entrées d'index par bloc**. Dans un index secondaire dense comme celui-ci, le nombre total d'entrées d'index ri est égal au nombre d'enregistrements dans le fichier de données, qui est de 300 000. Le nombre de blocs nécessaires pour l'index est donc bi =(ri/bfri) = (300 000/273) = **1 099 blocs.**

Une recherche binaire sur cet index secondaire nécessite (log2 bi) = (log21,099) = 11 accès bloc. Pour rechercher un enregistrement à l'aide de l'index, nous avons besoin d'un bloc d'accès supplémentaire au fichier de données pour un total de 11 + 1 = 12 blocs d'accès - une grande amélioration, mais légèrement pire que les 6 blocs d'accès requis pour l'index primaire. Cette différence est due au fait que l'indice primaire était non dense et donc plus court, avec seulement 28 blocs de longueur par opposition à l'indice dense de 1 099 blocs ici.

Indexes uni-niveau - revue

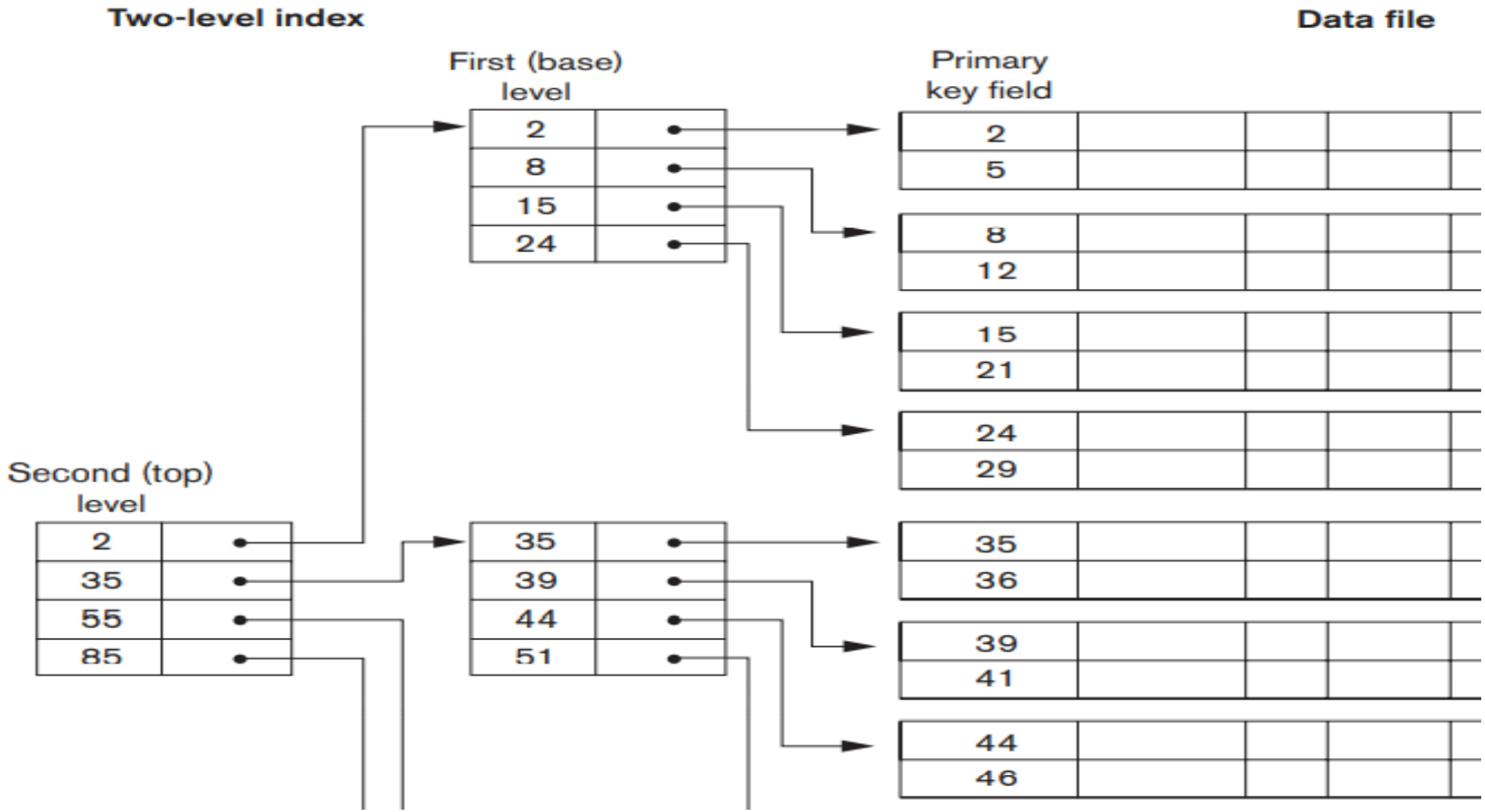
Type	Champ utilisé par index	Fichier non-ordonné par clé d'index	Nombre des entrées index	Dense	Accès par block
Primaire	clé	Non	Nombre de bloques	Non	Oui
Cluster	non-clé	Non	Nombre de champs index distincts	Non	Oui ou non
Secondaire (clé)	clé	Oui	Nombre d'enregistrements	Dense	Non
Secondaire (non clé)	non-clé	Oui	Nombre d'enregistrements avec des valeurs distinctes	Oui ou non	Non

Indexes multi-niveau

L'idée derrière un index à plusieurs niveaux est de réduire la partie de l'index que nous continuons à rechercher par $\mathbf{bfr_i}$, le facteur de blocage de l'index, qui est supérieur à 2. Par conséquent, l'espace de recherche est réduit beaucoup plus rapidement. La valeur $\mathbf{bfr_i}$ est appelée **fan-out de l'indice multiniveau**, et nous y ferons référence par le symbole **fo**. **Alors que nous divisons l'espace de recherche d'enregistrements en deux moitiés à chaque étape lors d'une recherche binaire, nous le divisons en n voies** à chaque étape de recherche en utilisant l'index multiniveau. La recherche d'un index à plusieurs niveaux nécessite environ ($\mathbf{\log_{fo} bi}$) accès au bloc, ce qui est un nombre sensiblement inférieur à celui d'une recherche binaire

Un index multiniveau considère le fichier d'index, le premier niveau (ou niveau de base) d'un index multiniveau, comme un fichier ordonné avec une valeur distincte pour chaque K(i).
Par conséquent, en considérant le fichier d'index de premier niveau comme un fichier de données triées, **nous pouvons créer un index primaire pour le premier niveau** ; cet indice au premier niveau est appelé le niveau secondaire de l'index. Étant donné que le deuxième niveau est un index primaire, nous pouvons utiliser des ancres de bloc afin que le deuxième niveau ait une entrée pour chaque bloc du premier niveau. Le facteur de blocage $\mathbf{bfr_i}$ pour le deuxième niveau - et pour tous les niveaux suivants - est le même

Indexes multi-niveau – exemple fan-out de 5



Indexes multi-niveau – mises a jour

Il reste des problèmes de traitement des insertions et des suppressions d'index, car tous les niveaux d'index sont des fichiers physiquement ordonnés.

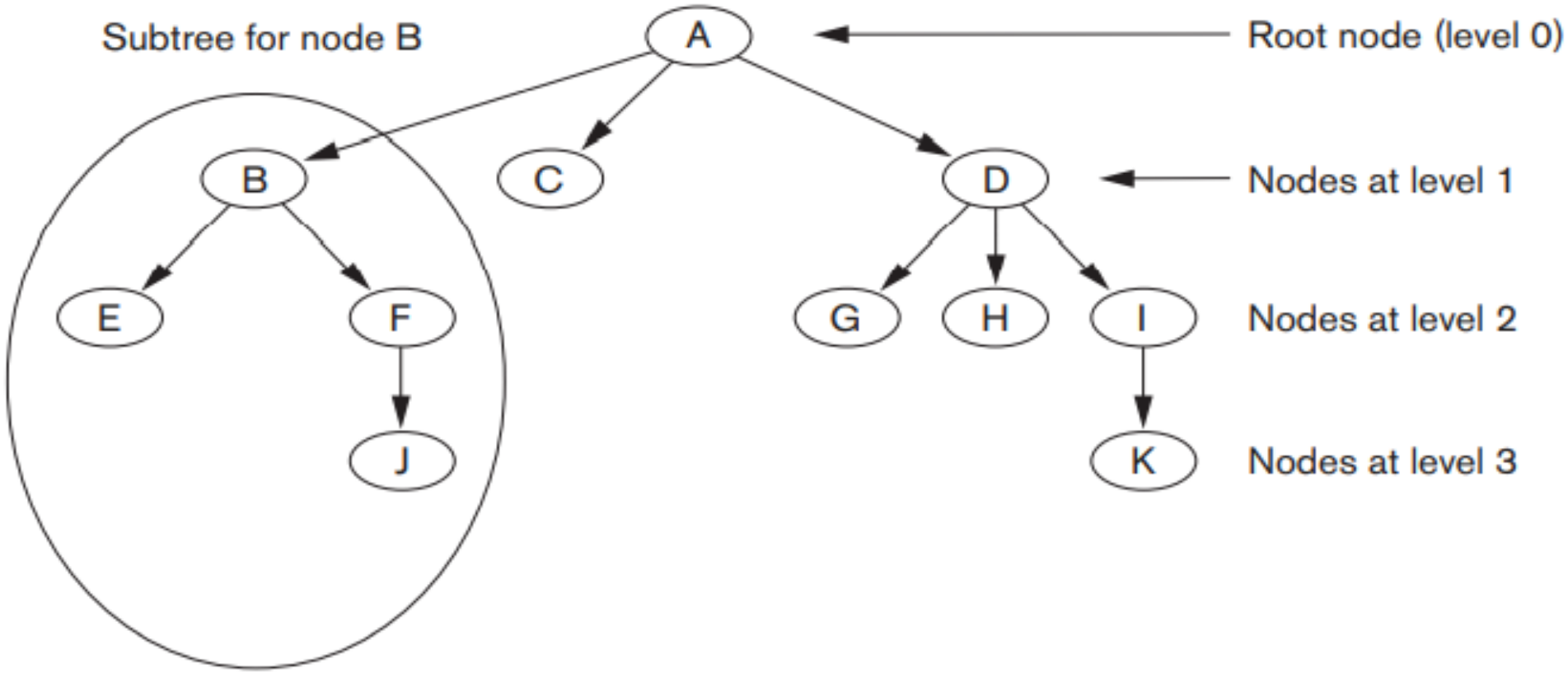
Pour conserver les avantages de l'utilisation de l'indexation multiniveau tout en réduisant les problèmes d'insertion et de suppression d'index, les concepteurs ont adopté un **index multiniveau** appelé **index multiniveau dynamique qui laisse un peu d'espace dans chacun de ses blocs pour insérer de nouvelles entrées** et utilise des algorithmes d'insertion/suppression appropriés pour créer et supprimer de nouveaux blocs d'index lorsque le fichier s'agrandit et se rétrécit.

Indexes B-tree

Les nœuds **B-tree** sont conservés entre 50 et 100 % pleins, et les pointeurs vers les blocs de données sont stockés à la fois dans les nœuds internes et les nœuds feuilles de la structure B-Tree.

Les arbres B+, une variante des arbres B dans laquelle **les pointeurs vers les blocs de données d'un fichier sont stockés uniquement dans les nœuds feuilles**, ce qui peut conduire à moins de niveaux et à des index de plus grande capacité. Dans les SGBD aujourd'hui, la structure commune utilisée pour l'indexation est les arbres B+.

Indexes B+-Tree
les nœuds leaf contiennent les pointeurs



(Nodes E, J, C, G, H, and K are leaf nodes of the tree)

Indexes composés – hachage partitionné

Il ne convient que pour les comparaisons d'égalité.

Dans le hachage partitionné, pour une clé composée de n composants, la fonction de hachage est conçue pour produire un résultat avec n adresses de hachage distinctes.

L'adresse du bucket est une concaténation de ces **n** adresses. Il est alors possible de rechercher la clé de recherche composite requise en recherchant les buckets appropriés.

L’index simple est une structure secondaire permettant d'accéder au fichier **en utilisant le hachage sur une clé de recherche** autre que celle utilisée pour l'organisation du fichier de données principal. Les entrées d'index sont du type **<K, Pr>** ou **<K, P>,** où **Pr** est un pointeur vers l'enregistrement contenant la clé, ou **P** est un pointeur vers le bloc contenant l'enregistrement pour cette clé.

Indexes composés – organisation grille

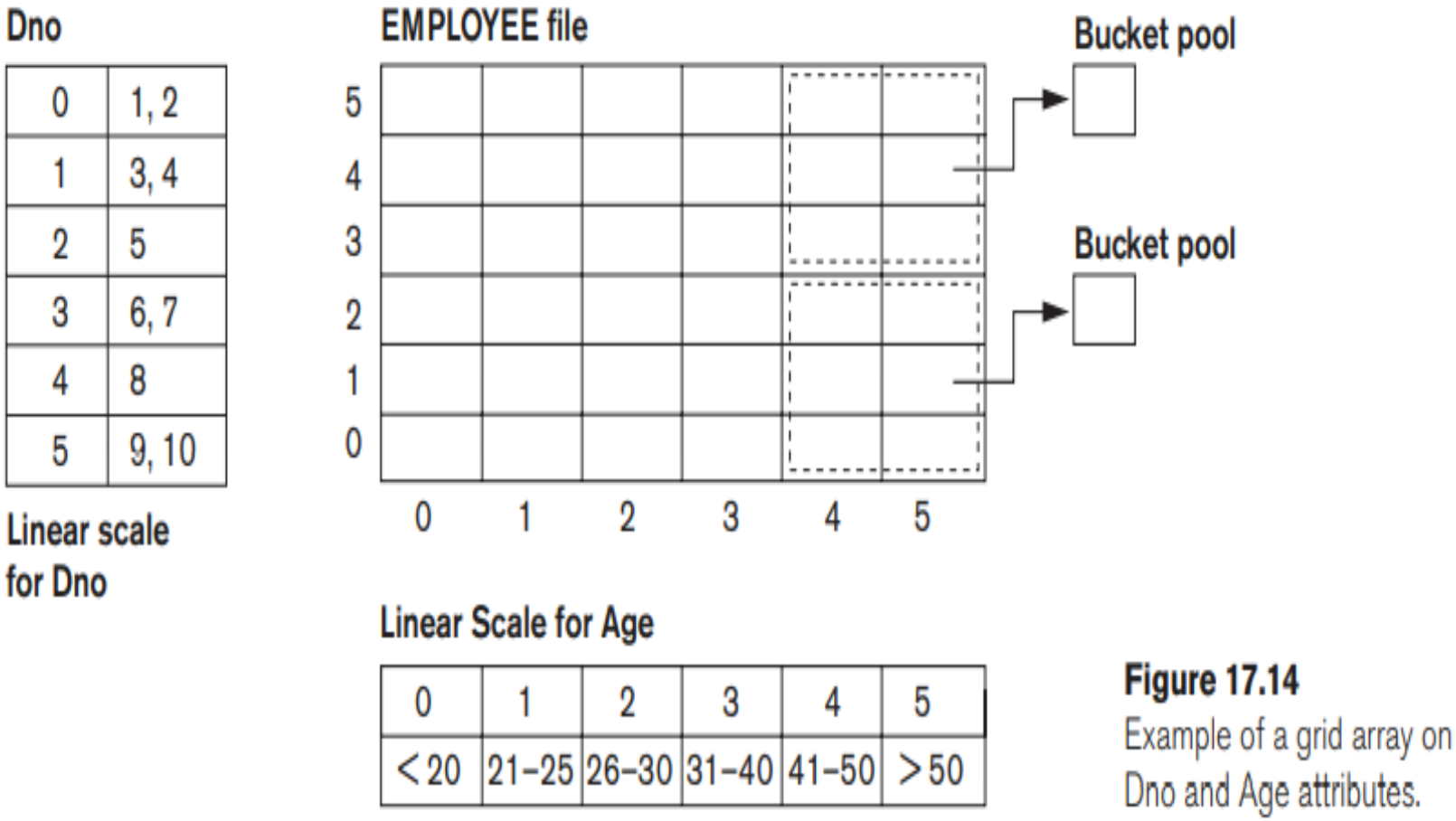


Figure 17.14
Example of a grid array on Dno and Age attributes.

Indexes bitmap

L'indexation bitmap est utilisée pour les relations qui contiennent un grand nombre de lignes avec un petit nombre de valeurs uniques.

Pour construire un index bitmap sur un ensemble d'enregistrements dans une relation, les enregistrements doivent être numérotés de 0 à n avec un identifiant (un identifiant d'enregistrement ou un identifiant de ligne) qui peut être mappé à une adresse physique **(rowid**) composée d'un numéro de bloc et d'un décalage d'enregistrements dans le bloc.

Un index bitmap est un tableau de bits. Ainsi, pour un champ donné, il existe un index bitmap (ou un vecteur) maintenu correspondant à **chaque valeur unique ROWID** dans la base de données.

Ex.: Pour trouver des employés avec Sexe = F et Code postal = 30022, nous croisons les bitmaps "01011001" et "01010010" donnant les Row_Ids 1 et 3.

Indexes type fonction

L'idée derrière l'indexation basée sur les fonctions est de créer un index tel que la valeur résultant de l'application d'une fonction sur un champ ou une collection de champs devienne la clé de l'index.

Les exemples suivants montrent comment créer et utiliser des index basés sur des fonctions.

DROP INDEX first_name_idx;

CREATE INDEX first_name_idx **ON** user_data (gender, **UPPER**(first_name), dob);

Attention aux collisions académiques et industrie

Taxonomie Microsoft

Les index clusterisés trient et stockent les lignes de données dans la table ou la vue en fonction de leurs valeurs de clé. Il s'agit des colonnes incluses dans la définition de l'index. Il ne peut y avoir qu'un seul index clusterisé par table, car les lignes de données elles-mêmes ne peuvent être stockées que dans un seul ordre.

Les **index non clusterisés** ont une structure distincte des lignes de données. Un index non-cluster contient les valeurs de clé d'index non-cluster et chaque entrée de valeur de clé a un pointeur vers la ligne de données qui contient la valeur de clé.

Types de données – predefinis – Numériques

Name	Storage Size	Description
smallint	2 bytes	small-range integer
integer	4 bytes	typical choice for integer
bigint	8 bytes	large-range integer
decimal	variable	user-specified precision, exact
numeric	variable	user-specified precision, exact
real	4 bytes	variable-precision, inexact
double precision	8 bytes	variable-precision, inexact
smallserial	2 bytes	small autoincrementing integer
serial	4 bytes	autoincrementing integer
bigserial	8 bytes	large autoincrementing integer

NUMERIC(précision, échelle)
NUMERIC(précision)

Ex.:
NUMERIC(12,2)

Autres valeurs possibles

- Infinity
- - Infinity
- NaN

https://www.postgresql.org/docs/current/datatype-numeric.html

Types de données – predefinis - Caracteres

Name	Description
`character varying(n)`, `varchar(n)`	variable-length with limit
`character(n)`, `char(n)`	fixed-length, blank padded
`text`	variable unlimited length

Ex.:
VARCHAR(20)
CHAR(15)
TEXT

https://www.postgresql.org/docs/current/datatype-character.html

Types de données – predefinis - Datetime

Name	Storage Size	Description
`timestamp [ (p) ] [ without time zone ]`	8 bytes	both date and time (no time zone)
`timestamp [ (p) ] with time zone`	8 bytes	both date and time, with time zone
`date`	4 bytes	date (no time of day)
`time [ (p) ] [ without time zone ]`	8 bytes	time of day (no date)
`time [ (p) ] with time zone`	12 bytes	time of day (no date), with time zone
`interval [ fields ] [ (p) ]`	16 bytes	time interval

Ex.:

```
04:05:06.787

2022-04-12 04:05:06
America/Montreal

TIMESTAMP '2004-10-19
10:23:54'
TIMESTAMP '2004-10-19
10:23:54-05'

Intervalles :
'3 days 2 months'
'3 4:05:06'
```

Fonctions compatibles SQL peuvent également être utilisées pour obtenir la valeur d'heure actuelle pour le type de données correspondant : CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, LOCALTIME, LOCALTIMESTAMP.

https://www.postgresql.org/docs/current/datatype-datetime.html

Types de données –binaires, booléennes, Json

Binaires 1-4 octets

Formats hex

SELECT '\xDEADBEEF';

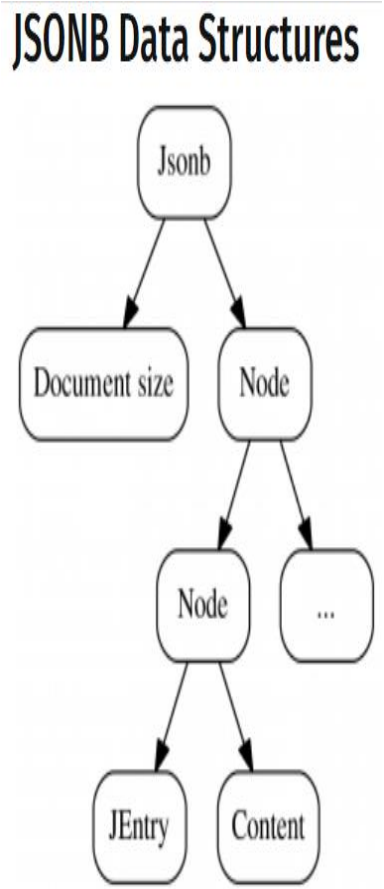
Formats escape – classique PostgreSQL

SELECT 'abc \153\154\155 \052\251\124'::bytea;

Boolean – 1 octet

Valeurs vraies : true, yes, on, 1

Valeurs fausses : false, no, off, 0



json stocke une copie exacte du texte d'entrée et les fonctions de traitement doivent valider ;

jsonb sont stockées dans un format binaire décomposé qui les rend plus lentes à saisir en raison de la surcharge de conversion supplémentaire, mais plus rapides à traiter, car aucune analyse n'est nécessaire. jsonb prend également en charge l'indexation;

Types de données – range

- int4range – Range of integer, int4multirange – corresponding Multirange
- int8range – Range of bigint, int8multirange – corresponding Multirange
- numrange – Range of numeric, nummultirange – corresponding Multirange
- tsrange – Range of timestamp without time zone, tsmultirange – corresponding Multirange
- tstzrange – Range of timestamp with time zone, tstzmultirange – corresponding Multirange
- daterange – Range of date, datemultirange – corresponding Multirange

```
CREATE TABLE tbl_SampleRange
(
        ID INT
        ,NumberRange INT4RANGE
        ,DateOnlyRange DATERANGE
        ,DateTimeRange TSRANGE
);

INSERT INTO tbl_SampleRange VALUES
(1,'[10,20)'::int4range,'[2015-06-
06,2016-08-08)','[2014-06-08
12:12:45,2016-07-06 14:12:08)')
,(2,'[40,50)'::int4range,'[2013-07-
07,2014-09-09)','[2015-05-05
11:08:40,2016-08-03 16:08:08)');
```

Types de données – composés

```
CREATE TYPE inventory_item AS (
    name            text,
    supplier_id     integer,
    price           numeric
);

CREATE TABLE on_hand (
    item      inventory_item,
    count     integer
);

INSERT INTO on_hand VALUES (ROW('numero bizarre', 42, 1.99), 1000);
```

PostgreSQL: Documentation: 14: 8.16. Composite Types

Séquences

La séquence est une fonctionnalité qui crée des nouvelles valeurs généralement uniques.

CREATE SEQUENCE serial **START** 101;

CREATE SEQUENCE cyclique **MINVALUE** value
MAXVALUE value **START WITH** value
INCREMENT BY value CYCLE
CACHE value;

ALTER SEQUENCE seq_name **INCREMENT BY** 10;

SELECT seq_name.nextval **FROM** dual;
SELECT seq_name.curval **FROM** dual;

Tables et Vues

```
CREATE TABLE my_first_table (
    colonne1 text,
    colonne1 integer
);
DROP TABLE my_first_table;

CREATE TABLE produits (
    product_no integer DEFAULT nextval('no_seq'),
    id_no SERIAL,
    name text,
    hauteur_cm numeric,
    hauteur_in numeric GENERATED ALWAYS AS (hauteur_cm / 2.54) STORED
    price numeric DEFAULT 9.99
);
```

Les vues sont simplement des texts SQLs représentés comme des objets dans une base de données.

CREATE [OR REPLACE][TEMP OR TEMPORARY] [RECURSIVE] VIEW view_name [(column_name [, ...])] [**WITH** (view_options_name [= view_options_value] [, ...])]

Contraintes

- **Vérifications (Check)**
- **Uniques**
- **Not Null**
- **Exclusions :** si deux lignes sont comparées sur la ou les colonnes ou expressions spécifiées à l'aide du ou des opérateurs spécifiés, au moins une de ces comparaisons d'opérateurs renverra faux ou null.

```
CREATE TABLE products (
    product_no integer UNIQUE ,
    name text NOT NULL,
    price numeric,
    CHECK (price > 0),
    discounted_price numeric,
    CHECK (discounted_price > 0),
    CONSTRAINT valid_discount CHECK (price
> discounted_price)
);
```

```
CREATE EXTENSION btree_gist;
CREATE TABLE example(
name varchar,
age integer,
EXCLUDE USING gist
(AGE WITH <>));

INSERT INTO example VALUES ('scott', '26');
INSERT INTO example VALUES ('scott', '27');
ERROR:  conflicting key value violates exclusion
constraint  "example_age_excl"
DETAIL: Key (age)=(27) ..
```

Clés

Primaires

```
CREATE TABLE products (
    product_no integer UNIQUE NOT NULL,
    name text,
    price numeric
);

CREATE TABLE example (
    a integer,
    b integer,
    c integer,
    PRIMARY KEY (a, c)
);
```

Référentielles

```
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);

CREATE TABLE products (
    product_no integer PRIMARY KEY,
    price numeric
);

CREATE TABLE t1 (
  a integer PRIMARY KEY,
  b integer,   c integer,
  FOREIGN KEY (b, c) REFERENCES other_table (c1, c2)
);
```

Indexes (1)

Type	Operateurs	Description	Exemple
B-Tree	< , <= , =, >=, > ,BETWEEN ,IN, IS NULL, IS NOT NULL, LIKE avec des exceptions	Les arbres B peuvent traiter des requêtes d'égalité et range sur des données qui peuvent être triées. Ils sont les plus populaires	**CREATE UNIQUE INDEX** title_idx **ON** films (title); **CREATE UNIQUE INDEX** title_idx **ON** films (title) **INCLUDE** (director, rating); **CREATE INDEX ON** films ((lower(title)));
Hash	=	Les index de hachage stockent un code de hachage 32 bits dérivé de la valeur de la colonne indexée. Ils sont plus petits et plus rapides en insertions et selects que les B-Tree mais ils ont beaucoup de restrictions	**DROP TABLE IF EXISTS** shorturl_hash; **CREATE TABLE** shorturl_hash (id serial **primary key**, **key** text **not null**, url text **not null**); **CREATE INDEX** shorturl_hash_hash_ix **ON** shorturl_hash **USING** hash(**key**);

Indexes (2)

Type	Operateurs	Description	Exemple
GIST	<< , &< , &> , >> , <<|, &<| , |&> , |>>, @>, <@, ~= , &&	Les index GiST (generalized search index sont une infrastructure avec plusieurs stratégies d'indexation. Utilisations possibles : ▪ Points (et autres entités géométriques) , recherche des plus proches voisins. ▪ Intervalles et contraintes d'exclusion. ▪ Recherche en texte intégral.	**CREATE TABLE** reservations(during TSRANGE); **CREATE INDEX** reservations **USING GIST**(during); **SELECT * FROM** RESERVATIONS **WHERE DURING** && '[2017-01-01, 2017-04-01)';

https://www.postgresql.org/docs/current/functions-range.html
https://www.postgresql.org/docs/current/functions-json.html

Indexes (3)

Type	Operateurs	Description	Exemple
SP-GIST	<< >> ~= <@ <<| |>>	GIST spatial, plus efficace pour les géométries sans chevauchement et stimule les recherches de distributions spatialement homogènes, grâce à son partitionnement spatial. Plus rapide pour une petite quantité de données	**CREATE TABLE** points(p point); **INSERT INTO** points(p) values (point '(1,1)'), (point '(3,2)'), (point '(6,3)'), (point '(5,5)'), (point '(7,8)'), (point '(8,6)');
GIN	<@ @> = &&	Les index GIN sont des "index inversés" conviennent aux valeurs de données contenant plusieurs valeurs et composants comme JSONB. Performance variable, pire que les indexes GIST, mais ils sont plus précis.	**CREATE INDEX** datatagsgin ON books **USING** gin (data->'tags');

https://www.postgresql.org/docs/current/functions-range.html
https://www.postgresql.org/docs/current/functions-json.html

Indexes (4)

Type	Operateurs	Description	Exemple
BRIN	<< >> ~= <@ <<| |>>	Block Range Index, stockent des résumés sur les valeurs stockées dans des plages de blocs physiques consécutives d'une table. Ils sont plus efficaces pour les colonnes dont les valeurs sont bien corrélées avec l'ordre physique de la table. Ils sont plus petits que les B-Tree et plus rapides en INSERTs et SELECTs.	**CREATE INDEX** testtab_date_brin_idx **ON** testtab **USING BRIN(**date**);**
BITMAP (Oracle)	< , <= , =, >=, > ,BETWEEN ,IN, IS NULL, IS NOT NULL, LIKE avec des exceptions	Un index bitmap est un type spécial d'index de base de données qui utilise des bitmaps ou des tableaux de bits. Dans un index bitmap, Oracle stocke un bitmap pour chaque clé d'index. Chaque clé d'index stocke des pointeurs vers plusieurs lignes.	**CREATE BITMAP INDEX** members_gender_i **ON** members(gender);

Sécurité BD

Access - suivez le principe du moindre privilège lorsque vous envisagez de configurer votre système ; c'est-à-dire, n'autoriser que l'accès nécessaire à la mise en œuvre d'un système opérationnel

- Rôles, utilisateurs, groupes , privilèges
- GRANT et REVOKE
- Sécurité basée sur les rangées et les colonnes d'une table (base de données virtuelles

Authentification – quel utilisateur se connecte

- **Externe** – un système externe contrôle l'authentification (GSSAPI, SSPI, LDAP, RADIUS)
- **OS** – le système d'opération contrôle l'authentification (PAM, Peer, Ident)
- **Interne** – la base de données contrôle l'authentification (Trust, Reject, md5, SCRAM, cert)

Audit

- Journalisation BD , Journalisation réseau , Journalisation OS , Journalisation applicative

Chiffrements de données

Masquage, Brouillage, Anonymisation, Cryptage de données

Sécurité BD – Accès (1)

- Rôles, utilisateurs, groupes

https://www.postgresql.org/docs/current/predefined-roles.html

CREATE ROLE admin WITH **CREATEDB CREATEROLE**;
CREATE ROLE miriam WITH LOGIN PASSWORD 'jw8s0F4' VALID UNTIL ‘2025-01-01';
CREATE USER david **WITH PASSWORD** 'jw8s0F4’;

Un rôle spécial est un "superutilisateur", qui peut outrepasser toutes les restrictions d'accès au sein de la base de données.

Sécurité BD – Accès (2)

- Privilèges

Il existe différents types de privilèges : **SELECT , INSERT , UPDATE , DELETE , TRUNCATE , REFERENCES , TRIGGER , CREATE , CONNECT , TEMPORARY , EXECUTE et USAGE** . Les privilèges applicables à un objet particulier varient selon le type d'objet (table, fonction, etc.).

Privilege	Abbreviation	Applicable Object Types
SELECT	r (“read”)	LARGE OBJECT, SEQUENCE, TABLE (and table-like objects), table column
INSERT	a (“append”)	TABLE, table column
UPDATE	w (“write”)	LARGE OBJECT, SEQUENCE, TABLE, table column
DELETE	d	TABLE
TRUNCATE	D	TABLE
REFERENCES	x	TABLE, table column
TRIGGER	t	TABLE
CREATE	C	DATABASE, SCHEMA, TABLESPACE
CONNECT	c	DATABASE
TEMPORARY	T	DATABASE
EXECUTE	X	FUNCTION, PROCEDURE
USAGE	U	DOMAIN, FOREIGN DATA WRAPPER, FOREIGN SERVER, LANGUAGE, SCHEMA, SEQUENCE, TYPE

Sécurité BD – Accès (3)

- **GRANT et REVOKE**

GRANT UPDATE ON accounts **TO** joe;
REVOKE ALL ON accounts **FROM PUBLIC;**
GRANT SELECT ON mytable **TO PUBLIC;**
GRANT SELECT, UPDATE, INSERT ON mytable **TO** admin;
GRANT SELECT (col1), UPDATE (col1) ON mytable **TO** miriam_rw;
GRANT ALL PRIVILEGES ON DATABASE n'implique pas le **SELECT** sur les tables

Lire toujours la documentation de la Sécurité

Sécurité BD – Accès (4)

La sécurité au niveau des rangées (RLS en abrégé) est une fonctionnalité de sécurité PostgreSQL qui permet aux administrateurs de base de données de définir des politiques pour contrôler la manière dont des lignes de données spécifiques s'affichent.

Les POLICYes limites les **INSERTs, SELECTS, UPDATEs et DELETEs**.

CREATE TABLE accounts (manager TEXT, company TEXT, contact_mail TEXT)
ALTER TABLE accounts **ENABLE ROW LEVEL SECURITY**
CREATE POLICY account_manager **TO** managers **USING** (manager = current_user)

Sécurité BD – Audit

L'extension d'audit PostgreSQL (pgAudit) fournit une journalisation détaillée de **l'audit des sessions et des objets** via la fonction de journalisation standard de PostgreSQL.

CREATE EXTENSION pgaudit;

Il est possible d'auditer :

- **READ :** SELECT et COPY
- **WRITE :** INSERT, UPDATE, DELETE, TRUNCATE et COPY
- **FUNCTION :** Appels de fonctions
- **ROLE :** Déclarations DCL relatives aux rôles et privilèges : GRANT, REVOKE, CREATE/ALTER/DROP ROLE.
- **DDL :** Tous les DDL qui ne sont pas inclus dans la classe ROLE.

Sécurité BD – Chiffrements

- **Chiffrement dans les communications BD**

- **Chiffrement de schéma interne**
  - Chiffrement logique sur les colonnes d'une table
  - Chiffrement sur les partitions d'une table
  - Chiffrement physiques de fichiers BD

- **Chiffrement dans la présentation de données**
  - Masquage de colonnes , rangées
  - Brouillage, impossible de récupérer la vérité
  - Jetons , il est possible de récupérer la vérité