Patrice BOURSIER

Professeur, Univ. de La Rochelle

patrice.boursier@univ-lr.fr

Bases de Données

Notes de cours

SOMMAIRE

1. INTRODUCTION

1.1. Objectifs

- Définition : base de données (BD) = ensemble de **données** accessibles et exploitables au moyen d'un ensemble de **programmes**.
- Définition : système de gestion de bases de données (SGBD) = outil (logiciel) permettant d'accéder à des BD.
- Exemples de SGBD :
  - professionnels relationnels : Oracle, Sybase, Informix, ...,
  - professionnels orientés-objets : Versant, Objectstore, O2, ...
  - personnels : Access, Paradox, 4D, ...,
- Exemples d'utilisations :
  - consultation (en local ou à distance[1]) des données relatives aux produits d'une entreprise,
  - consultation à distance des cours de la bourse,
  - facturation à partir des données relatives aux commandes-clients et aux caractéristiques de produits, ...
- Notions de **données** vs. **informations** : donnée = information codée sur un support (définition personnelle non officielle)
- Notion de **système d'information** : plus générale que celle de BD. Inclut la description des flux de données, les aspects organisationnels, ...
- Objectifs des SGBD :
  - exploitation de gros volumes de données => structures de données et méthodes d'accès efficaces,
  - exploitation par différents types d'utilisateurs => différents outils d'accès ou interfaces-utilisateurs,
  - gestion de données sensibles => sécurité et fiabilité des outils,
  - aspect multi-utilisateurs => mécanismes de protection[2],

1 intro. Internet, Intranet

2 cf. cours Système

1.2. Justification historique

- Avant les SGBD : écriture de programmes par des programmeurs d'application utilisant le système de gestion de fichiers[3] pour gérer et exploiter les données

  ➔ risques liés au manque de sécurité + multiplication des efforts (programmes similaires écrits dans différents services pour des besoins proches).

- Conséquences :
  - **redondances** : fichiers contenant les mêmes données, mais utilisées par des personnes différentes,
  - risque d'**incohérences** : du fait des redondances et des MAJ non centralisées (ex: adresse d'un fournisseur),
  - **intégrité** des données : respect de contraintes qui peuvent être programmées (ex: contrôles sur date de naissance, sur code pôstal, numéro de tél., ...),
  - pbs liés à la **sécurité** : utilisateurs de différents niveaux d'expérience et avec différents droits d'accès => mots de passe,
  - pbs liés au **partage** des données : accès en lecture / écriture.

1.3. Indépendance données / programmes

L'objectif premier des SGBD est d'assurer cette indépendance, en libérant les programmeurs et les utilisateurs en général de la connaissance précise de la façon dont les données sont structurées.

1.4. Niveaux d'abstraction

On a coutume de distinguer plusieurs niveaux de représentation ou d'abstraction pour les bases de données et les systèmes d'information de manière plus générale :

- le niveau externe (utilisateur) -> **vues**
- le niveau conceptuel (concepteur, administrateur) -> **modèles de données**
- le niveau interne (stockage) -> **structures de données** (fichiers, index)

1.5. Modèles de données

On distingue généralement deux catégories de modèles de données :

- les modèles orientés information -> définition du **schéma conceptuel**

  ➔ modèle entité-association et ses dérivés (MERISE, ...)

- les modèles orientés données -> mise-en-œuvre du SGBD

[3] sous-ensemble du système d'exploitation

➔ modèle relationnel, modèle hiérarchique, modèle réseau/CODASYL

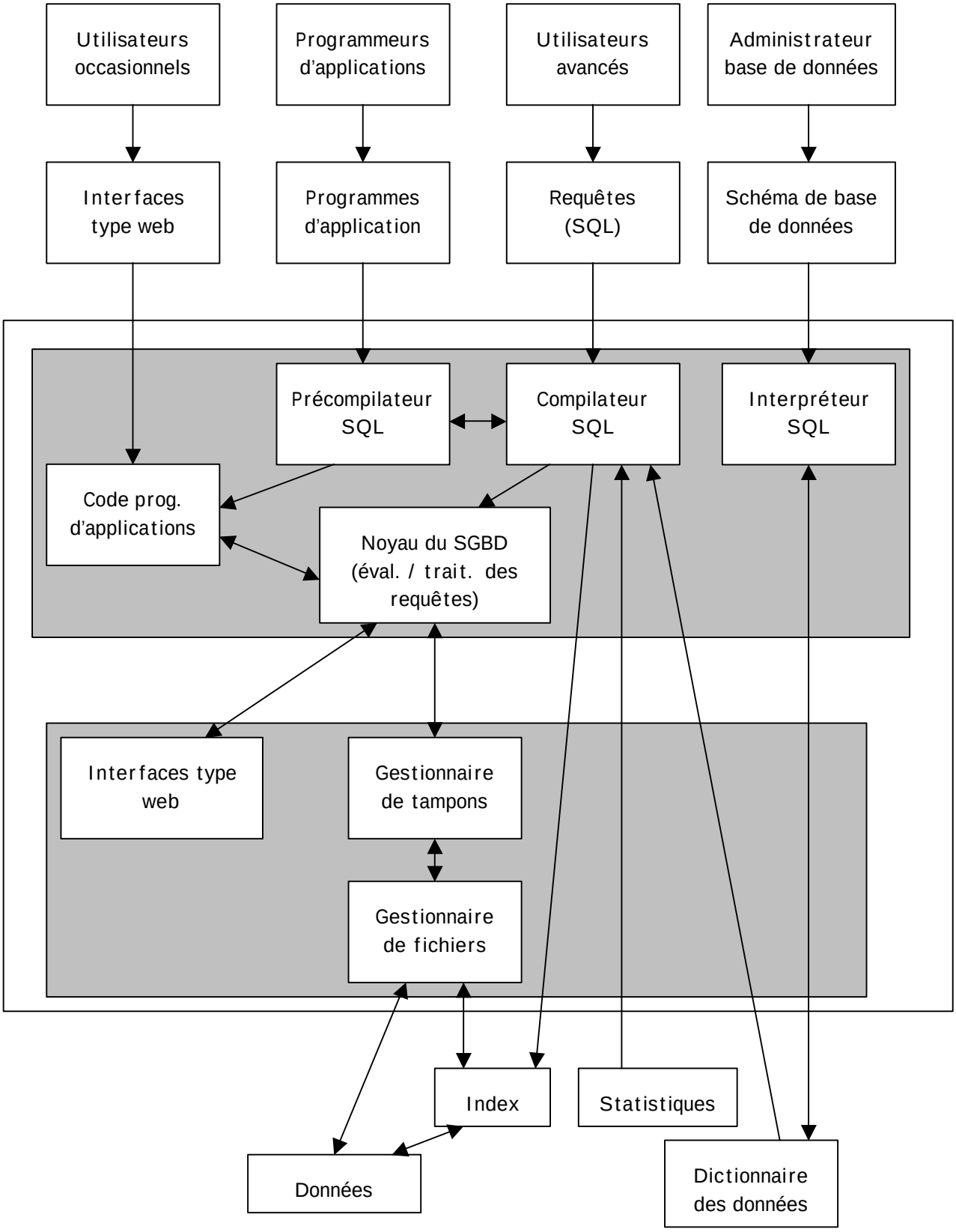
1.6. Langages

- 2 types de langages :
  - langages procéduraux
    ➔ décrivent à la fois ce que l'utilisateur souhaite et l'algorithme de résolution du problème : "quoi et comment"
  - langages déclaratifs (ou assertionnels)
    ➔ se limitent à décrire la demande de l'utilisateur : "quoi"
- 2 types de fonctionnalités dans les BD
  - langage de définition (LDD)
    ➔ définition du **dictionnaire des données** (méta-données)
  - langage de manipulation (LMD)
    ➔ consultation + mise à jour (insertion, suppression, modification)

1.7. Fonctions liées à l'exploitation des SGBD

- Administrateur de bases de données
  ➔ analyse fonctionnelle
  ➔ définition du schéma conceptuel
  ➔ choix des méthodes de stockage et d'accès
  ➔ modification de l'organisation conceptuelle / physique
  ➔ gestion des droits d'accès
  ➔ spécification des contraintes d'intégrité
- Classes d'utilisateurs
  ➔ utilisateurs occasionnels (programmes d'application)
  ➔ utilisateurs experts (SQL)
  ➔ programmeurs d'applications (SQL + langage de programmation "hôte")

1.8. Fonctionnalités et organisation générale



2. MODELISATION CONCEPTUELLE

2.1. Introduction

Objectif = représentation symbolique et réduite de la réalité

➔ représentation **logique** (vs. physique)

Les modèles conceptuels de type **entité-association** (E/A) tels que MERISE sont progressivement remplacés par des modèles s'inspirant des méthodes de conception de type **orienté-objet**. On qualifie aussi ces modèles de langages de modélisation (cf. UML : Unified Modeling Language).

2.2. Entité

Définition : entité = représentation logique d'un individu (au sens large, c'est-à-dire élément d'un ensemble) ou "objet".

Une entité correspond également à une classe d'

Une entité est caractérisée par son **unicité** => identifiant unique ("**clé**"). Ex: no. SS, no. client, no. produit.

Une entité peut être représentée par un ensemble d'**attributs**[4]. Ex: nom d'une personne, âge, adresse, nom d'un produit, prix unitaire, quantité en stock, ...

Chaque attribut peut prendre un certain nombre de **valeurs** -> **domaine** de l'attribut[5].

Une entité du monde réel est ainsi représentée par un ensemble d'attributs **instanciés** à l'aide de valeurs.

Une entité peut être considérée comme **dépendante** si son existence est conditionnée par l'existence d'une autre ou d'autres entités. Ex: si un produit n'est livré que par un seul fournisseur et que ce fournisseur disparait, le produit va lui aussi disparaitre. On parle aussi d'entités dominantes / dominées.

2.3. Association

Une association relie des classes d'entités entre elles.

Ex: association reliant : (i) des commandes caractérisées par une date de commande, le numéro d'entification du client, les numéros de produits commandés ainsi que les quantités, avec (ii) des produits vendus par l'entreprise et

[4] la clé en fait partie.

[5] à rapprocher de la notionde type en programmation (types étendus)

caractérisées par un numéro de produit, un libellé, un prix unitaire et une quantité en stock.

Une association relie en général deux classes d'entités (ass. binaire), mais elle peut aussi relier plusieurs classes d'entités (ass. n-aires).

Les entités jouent des **rôles** différents dans une association. Ex: un client *passe* une commande, la commande est passée[6] par un client (autre classe d'entités), le produit est *référencé* dans la commande, la commande *référence* certais produits, ...

Au sein d'une même classe, une ou plusieurs entités peuvent être mises en relation avec une ou plusieurs entités d'une autre classe

-> associations de type 1-1, 1-n ou n-m

-> *cardinalité* d'une association

Exemple : une commande peut concerner plusieurs produits, mais un produit peut aussi faire partie de plusieurs commandes -> assoc. n-m.

Attention aux *interprétations* de la réalité pouvant conduire à des représentations différentes et parfois erronées -> importance de la phase d'enquête et d'analyse auprès des utilisateurs.

2.4. Identificateurs et clés

Rappel : chaque entité est caractérisée par un identicateur (ou identifiant) **unique**, qu'on appelle aussi **clé** (ou clé primaire) d'accès à l'entité. Lorsqu'une entité possède plusieurs identifiants uniques, la clé peut être choisie parmi ces identifiants qui sont considérés comme un ensemble de **clés candidates**.

Cet identifiant peut être constitué par un seul attribut de l'entité (ex. no. SS), mais il peut être nécessaire de combiner plusieurs attributs afin de constituer la clé. On parlera alors de **clé composée**. Ex: pour des personnes dont on ne connait pas le no. SS, on peut combiner nom, prénom et code postal ou année de naissance afin de réduire au maximum les risques de combinaisons équivalentes (ou "doublons").

On pourra aussi affecter à une association un identifiant unique qui sera obtenu par combinaison des identifiants des entités qui y participent.

Dans certains cas, il peut s'avérer impossible de constituer un identifiant unique. Dans ce cas, on qualifiera l'entité d'*entité faible* (vs. entité forte). Cette notion est à rapprocher de la notion précédente d'entité dépendante ou dominée : une entité faible est une entité dominée.

[6] remarquer la dualité présent / passé

2.5. Diagrammes entités-associations

Classes d'entités -> rectangles.

Atributs -> ellipses

Associations -> losanges.

Exemples : Produits / Fournisseurs / Clients, Comptes bancaires / Clients / Agences, Etudiants / Cours / Profs.

2.6. Spécialisation / généralisation

Certaines classes d'entités peuvent se spécialiser en sous-classes, par ex. les comptes bancaires se spécialisent en comptes-chèques et en comptes d'épargne.

Inversement, les comptes-chèques et comptes d'épargne sont des sous-ensembles de la classe plus générale des comptes bancaires.

-> assoc. "Is-a"

Notion d'héritage : une entité spécialisée *hérite* des attributs de l'entité généralisée.

3. MODELE RELATIONNEL

3.1. Introduction

- Introduit par E.F. Codd en 1970
- Premiers systèmes commercialisés à la fin des années 70, systèmes efficaces au début des années 80
- Modèle fondé sur la **théorie des ensembles** et la notion de **relation** (bases mathématiques)
- Définition d'une relation : R (A_1 : D_1, A_2 : D_2, … A_n : D_n), avec :

  R = nom de la relation

  A_i = noms des attributs

  D_i = domaines de définition des attributs (valeurs possibles)

  n = cardinalité de la relation

Exemple : PERSONNE (Nom : char(20), Prénom : char(20), Age : integer, Adresse : varchar(50), CP : integer, Ville : char(20))

Ce qui est écrit ci-dessus constitue en fait le **schéma de la relation** PERSONNE. La **relation** PERSONNE est représentée sous la forme d'une table :

Nom	Prénom	Age	Adresse	CP	Ville
Dupont	Pierre	50	7, rue du Port	17000	La Rochelle
Martin	Alain	33	4, place de la Gare	87000	Limoges

- Une ligne de la table constitue un élément de la relation, ou **n-uplet**. Elle représente aussi une personne, instance de la relation PERSONNE. D'un point de vue logique (mathématique), il s'agit d'un prédicat qui met en relation les attributs de la relation.
- Il n'y a pas d'ordre sur les lignes (ni sur les colonnes) dans une table / relation.
- Il n'y a pas non plus d'information sur l'organisation physique (stockage des données) qui est de ce fait cachée à l'utilisateur.

3.2. Du modèle entité-association au modèle relationnel

- 1 entité se traduit par 1 relation, qui contient tous les attributs de l'entité

- 1 association se traduit par 1 relation, qui contient tous les attributs de l'entité ainsi que les attributs de l'association

Exemple : schéma entité-association « Cours / Etudiants / Profs »

Il se traduit par le **schéma relationnel de BD** (ensemble de schémas de relation) suivant :

ETUDIANT (Num_Etudiant **integer**, Nom **char(20)**, Adresse **varchar(50)**)

COURS (Num_Cours **integer**, Nom **char(20)**)

PROFS (Num_Prof **integer**, Nom **char(20)**, Adresse **varchar(50)**)

COURS_SUIVIS (Num_Etudiant **integer**, Num_Cours **integer**)

COURS_ENSEIGNES (Num_Prof **integer**, Num_Cours **integer**)

3.3. Langages de manipulation

Deux classes de langages s'appliquent au modèle relationnel :

- l'algèbre relationnelle, qui est un langage de type procédural,
- les langages de calcul relationnel , qui sont des langages de type déclaratif (ou assertionnel), avec d'une part le calcul relationnel à variables domaines et d'autre part le calcul relationnel à variables n-uplets.

Le langage SQL ("Structured Query Language") est le langage d'interrogation de référence pour les utilisateurs de SGBD relationnels (cf. chapitre suivant).

4. LANGAGES DE REQUETES / SQL

4.1. Introduction

- SQL = Structured Query Language
- Langage de requêtes standard pour l'interrogation de bases de données relationnelles (SQL-1 en 1989, puis SQL-2 en 1992, SQL-3 ?)
- Développé à l'origine pour le protype de SGBD recherche d'IBM SYSTEM/R, qui a débouché sur les produits commerciaux SQL/DS et DB-2
- Mélange d'algèbre relationnelle et de calcul relationnel à variables n-uplets

4.2. Opérations sur tables

- Définition de schémas de relations (et création des tables correspondantes) :

  create table nom_relation (nom_attribut_1 type_attribut_1, …)
- Suppression de tables :

  drop table nom_relation
- Modification de tables (ajout de colonne) :

  alter table nom_relation **add** nom_attribut

4.3. Opérations sur n-uplets

- Insertion de n-uplets dans une table :

  insert into nom_relation **values** (valeur_attribut_1, …, valeur_attribut_n)
- Suppression de n-uplets dans une table :

  delete from nom_relation [**where** condition]
- Modification de n-uplets dans une table :

  update nom_relation **set** nom_attribut = valeur_attribut [**where** condition]

4.4. Opération de consultation

select nom_attribut_1, …, nom_attribut_n

from nom_relation_1, …, nom_relation_m

where condition_1, …, condition_p

Quelques « trucs » à savoir :

- pour voir tous les attributs d'une relation : **select** * **from** nom_table

- élimination des doubles : **select distinct** nom_attribut …
- ordonnancement des résultats : **order by** nom_attribut (à la fin de la requête)
- opérateurs arithmétiques : = != > >= < <=
- opérateurs logiques : **and**, **or**, **not**
- test entre valeurs : nom_attribut **between** val_attr_1 **and** val_attr_2
- appartenance d'une valeur d'attribut à un ensemble : [**not**] **in**, **any**, **all**
- fonctions agrégat : **avg**, **sum**, **min**, **max**, **count**
- utilisation des fonctions agrégat avec groupage : **group by**, **having**

Attention : même si le langage SQL est normalisé, chaque SGBD a des particularités syntaxiques => voir la documentation du système utilisé !

5. SQL - DROITS D'ACCES ET VUES

5.1. Introduction

- Nécessité de mécanismes de protection et de sécurité dans les SGBD à différents niveaux :
  a) contrôle des utilisateurs souhaitant accéder à une BD
  b) contrôle de l'accès aux données (lecture, écriture)
  c) contrôle de l'intégrité des données, et de la validité des opérations de MAJ
- Deux classes de droits d'accès aux données :
  a) consultation
  b) MAJ -> insertion, suppression, modification
- Exemples de droits d'accès sur une BD :

  Relation PERSONNEL (Id, Nom, Adresse, Service, Salaire)

  a) Dupont possède tous les droits d'accès (consultation, MAJ) sur la relation PERSONNEL
  b) Dupont ne possède aucun droit
  c) Dupont ne peut que lire que l'information le concernant, et il ne peut pas la modifier
  d) Dupont ne peut que lire que l'information le concernant, et il peut modifier son adresse
  e) Dupont ne peut que lire que l'information le concernant, et il peut modifier son salaire s'il est > 75 000 F
  f) Dupont ne peut que lire que l'information le concernant, et il peut modifier son salaire s'il est responsable du service (information présente dans une autre table)
- Pb : le schéma de la BD peut évoluer au fil du temps (MAJ des schémas de relation)

  => le mécanisme de contrôle des droits d'accès doit en tenir compte

5.2. Mécanismes d'octroi et d'annulation des droits d'accès

- Deux commandes pour l'administrateur de la base de données : **grant** (octroi) et **revoke** (annulation)

- Les droits d'accès sot stockés dans le dictionnaire des données (méta-base). Ils sont utilisés par le SGBD pour vérifier la validité des accès à une BD
- Syntaxe en SQL-2 :
  - **usage** -> accès au domaine de définition d'un attribut (ou d'un ensemble d'attributs)
  - **select** -> accès en lecture (consultation)
  - **insert (col)** -> insertion de valeurs dans la colonne spécifiée
  - **insert** -> insertion de valeurs dans toutes les colonnes d'une table
  - **update (col)** -> modification de valeurs dans la colonne spécifiée
  - **update** -> modification de valeurs dans toutes les colonnes d'une table
  - **delete** -> suppresion de n-uplets
  - **references** (col) et **references** -> possibilité de faire référence à certaines colonnes ou à toutes les colonnes d'une table dans des contraintes d'intégrité
  - **all** -> tous les droits
- Forme générale de la commande **grant** :

  grant droit **on** objet **to** utilisateur [**with grant option**]

  objet = table(s), vue(s), colonne(s)

  utilisateur = 1 ou plusieurs utilisateurs, ou **public** pour désigner tous les utilisateurs

  with grant option = transmission des droits à d'autres utilisateurs
- Exemple d'utilisation sur la table PERSONNEL
  a) le directeur possède tous les droits, et il peut les transmettre

     grant all on table PERSONNEL **to** 'directeur' **with grant option**

  b) la secrétaire peut seulement insérer des n-uplets

     grant insert on table PERSONNEL **to** 'secrétaire'

  c) le DRH peut augmenter les salaires

     grant select, update (Salaire) **on table** PERSONNEL **to** 'DRH'
- Forme générale de la commande **revoke** :

  revoke droit **on** objet **from** utilisateur

  exemple : **revoke select on table** PERSONNEL **from** 'Dupont'

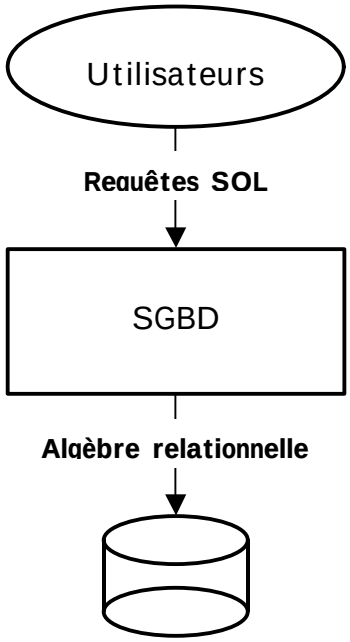
5.3. Définition des vues

- Rappel : Les tables …

6. ALGEBRE RELATIONNELLE

6.1. Introduction

- Les requêtes SQL soumises par l'utilisateur sont traduites par le SGBD en opérations de l'algèbre relationnelle. L'algèbre relationnelle est le langage de manipulation qu'utilise le SGBD pour effectuer des opérations sur les relations (tables).



6.2. Opérateurs

- Deux types d'opérateurs, unaires (opérant sur une relation) et binaires (opérant sur deux relations).
- Cinq opérateurs de base : sélection (σ), projection (π), union ($\cup$), différence (-) et produit cartésien (x).
- Autres opérateurs déduits (pour faciliter l'expression des requêtes) : intersection ($\cap$), jointure ($\triangleright\triangleleft$) et division ($\div$).
- Opérateurs déduits à partir des opérateurs de base :
  - $r \cap s = r - (r - s)$
  - $r \triangleright\triangleleft s = \sigma (r \times s)$
  - $r \div s = \pi_{R-S} (r) - \pi_{R-S} ((\pi_{R-S} (r) \times s) - r)$

6.3. Définition formelle des opérateurs

- Union ($r \cup s$) : l'union de 2 relations r et s (de même schéma) est une relation t (de même schéma) contenant l'ensemble des n-uplets appartenant à r ou à s ou aux 2 relations
- Différence ($r - s$) : la différence de 2 relations r et s (de même schéma) est une relation t (de même schéma) contenant l'ensemble des n-uplets appartenant à r et n'appartenant pas à s
- Intersection ($r \cap s$) : l'intersection de 2 relations r et s (de même schéma) est une relation t (de même schéma) contenant l'ensemble des n-uplets appartenant à la fois à r et à s
- Projection ($r \pi s$) : la projection d'une relation r de schéma R (A_1, A_2, …, A_n) sur les attributs A_{i1}, A_{i2}, …, A_{ip} est une relation r' de schéma R' (A_{i1}, A_{i2}, …, A_{ip}) dont les n-uplets sont obtenus par élimination des valeurs des attributs de R n'appartenant pas à R' et par suppression des n-uplets en double
- Sélection ($r \sigma s$) : la sélection sur la relation r par une qualification Q est une relation r' de même schéma dont les n-uplets sont ceux de r satisfaisant Q
- Produit cartésien ($r \times s$) : le produit cartésien de 2 relations r et s (de schéma quelconque) est une relation t ayant pour attributs la concaténation de ceux de r et de s, et dont les n-uplets sont toutes les cocaténations d'un n-uplet de r à un n-uplet de s
- Jointure ($r \triangleright\triangleleft s$) : la jointure de 2 relations r et s (de schéma quelconque) suivant une qualification multi-attributs Q est une relation t contenant l'ensemble des n-uplets du produits cartésien r x s satisfaisant Q
- Division ($r \div s$) : le quotient de la relation r de schéma R (A_1, A_2, …, A_n) par la sous-relation s de schéma S (A_{p+1}, …, A_n) est la relation q de schéma Q (A_1, …, A_p) formée de tous les n-uplets qui, concaténés à chacun des n-uplets de s, donnent toujours un n-uplet de r

Exemples :

Soit le schéma de relation FOURNISSEUR (Nom_Fournisseur, Adresse, Produit, Prix).

7. CALCUL RELATIONNEL

7.1. Introduction

8. NORMALISATION

8.1. Introduction

L'objectif de la normalisation est de construire un schéma de base de données cohérent et possédant certaines propriétés vérifiées par la satisfaction de **formes normales**.

Pour une application spécifique, il est en effet possible de proposer plusieurs schémas. Les questions qui se posent alors sont les suivantes :

a) qu'est-ce qu'un bon schéma ?

b) quel schéma choisir ?

Un mauvais schéma défini lors de la phase de conception peut conduire à un certain nombre d'anomalies pendant la phase d'exploitation de la base :

- des **redondances** d'information,
- des **anomalies** lors des opérations de **mise à jour** (insertions, suppressions, modifications).

Exemple :

Soit le schéma de relation FOURNISSEUR (Nom_Fournisseur, Adresse, Produit, Prix).

Une relation (table) correspondant à ce schéma pourra éventuellement contenir plusieurs produits pour un même fournisseur. Dans ce cas, l'adresse du fournisseur sera dupliquée dans chaque n-uplet (redondance).

Si on souhaite modifier l'adresse d'un fournisseur, il faudra rechercher etmettre à jour tous les n-uplets correspondant à ce fournisseur.

Si on insère un nouveau produit pour un fournisseur déjà référencé, il faudra vérifier que l'adresse est identique.

Si on veut supprimer un fournisseur, il faudra retrouver et supprimer tous les n-uplets correspondant à ce fournisseur (pour différents produits) dans la table.

Ces anomalies n'apparaitront pas si on décompose le schéma initial de base de données.

Par contre, la décomposition d'un schéma relationnel au cours de la normalisation risque d'entrainer une dégradation des performances, du fait des opérations de jointure nécessaires.

Les 3 premières formes normales ont été proposées par E.F. Codd ("inventeur" du modèle relationnel) en 1972. La forme normale dite de Boyce-Codd a été proposée en 1974. Les 4ème (1977) et 5ème (1979) formes normales ont été proposées ensuite par Fagin, mais elles ne concernent que des cas rares et très spécifiques.

Les formes normales s'appuient sur les **dépendances fonctionnelles** entre attributs d'un schéma de base de données.

8.2. Dépendances fonctionnelles

Définition : un attribut (ou un groupe d'attributs) B est dit "fonctionnellement dépendant" d'un attribut (ou d'un groupe d'attributs) A si :

a1 = a2 => b1 = b2,

a1, a2, b1, b2 étant des réalisations (valeurs) des attributs A et B dans des n-uplets de la base de données.

On dit alors que **A "détermine" B**, et on note **A -> B**.

Exemple :

Soit le schéma de relation PERSONNE (No_SS, Nom, Adresse, Age, Profession).

Les dépendances fonctionnelles qui s'appliquent sur ce schéma de relation sont les suivantes :

No_SS -> Nom, No_SS -> Adresse, No_SS -> Age, No_SS -> Profession.

On pourra aussi écrire :

No_SS -> Nom Adresse Age Profession.

L'attribut No_SS détermine tous les attributs du schéma de relation. Il s'agit d'une **propriété de la clé** d'une schéma de relation.

Exercice :

Soit la relation suivante **r** de schéma **R** (A, B, C, D, E).

A	B	C	D	E
a1	b1	c1	d1	e1
a1	b2	c2	d2	e1
a2	b1	c3	d3	e1
a2	b1	c4	d3	e1
a3	b2	c5	d1	e1

Les dépendances fonctionnelles satisfaites par R sont les suivantes :

A -> E ; B -> E ; C -> ABDE ; D -> E ; AB -> D ; AD -> B ; BD -> A.

8.3. Axiomes d'Armstrong et couverture minimale

A partir d'un ensemble F de dépendances fonctionnelles entre les attributs d'un schéma de relation R, on peut en déduire d'autres à partir des trois propriétés suivantes :

a) transitivité : si X -> Y, et Y -> Z, alors X -> Z,

b) augmentation : si X -> Y, alors XZ -> Y pour tout groupe Z d'attributs appartenant au schéma de relation,

c) réflexivité : si X contient Y, alors X -> Y.

A partir de ces trois axiomes de base, on peut déduire d'autres règles :

d) union : si X -> Y et Y -> Z, alors X -> YZ,

e) pseudo-transitivité : si X -> Y et WY -> Z, alors WX -> Z,

f) décomposition : si X -> Y et Z contenu dans Y, alors X -> Z.

La **fermeture d'un ensemble d'attributs** X, notée **$(X)^+$** représente l'ensemble des attributs de R qui peuvent être déduits de X à partir d'une famille de dépendances fonctionnelles en appliquant les axiomes d'Armstrong. Ainsi, Y sera inclus dans $(X)^+$ ssi X -> Y.

Calcul de la fermeture d'un ensemble d'attributs :

a) initialiser $(X)^+$ à X,

b) trouver une DF de F possédant en partie gauche des attributs inclus dans $(X)^+$,

c) ajouter dans $(X)^+$ les attributs placés en partie droite de la DF,

d) répéter les étapes b) et c) jusqu'à ce que $(X)^+$ n'évolue plus.

Exercice :

Soit F = { A -> D ; AB -> E ; BI -> E ; CD -> I ; E -> C }.

a) Calculer la fermeture, sous F, de AE.

Solution : au départ, $(AE)^+$ = AE,

A -> D permet d'ajouter D : $(AE)^+$ = AED,

E -> C permet d'ajouter C : $(AE)^+$ = AEDC,

CD -> I permet d'ajouter I : $(AE)^+$ = AEDCI.

b) Calculer la fermeture, sous F, de BE.

Solution : au départ, $(BE)^+$ = BE,

E -> C permet d'ajouter C : $(BE)^+$ = BEC.

Exercice :

Soit F = { AB -> C ; B -> D ; CD -> E ; CE -> GH ; G -> A }.

En utilisant la notion de fermeture d'un ensemble d'attributs, montrer que :

a) AB -> E,

Solution : B -> D |= AB -> D par augmentation

AB -> C et AB -> D |= AB -> CD par union

AB -> CD et CD -> E |= AB -> E par transitivité

b) BG -> C,

Solution : G -> A |= BG -> A par augmentation

BG -> BG |= BG -> B par projection

BG -> A et BG -> B |= BG -> AB par union

BG -> AB et AB -> C |= BG -> C par transitivité

c) AB -> G.

Solution : AB -> E et AB -> C |= AB -> CE par additivité

AB -> CE et CE -> GH |= AB -> GH par transitivité

AB -> GH |= AB -> G par projection

Inversement, on peut simplifier F en supprimant les dépendances fonctionnelles redondantes, c'est à dire celles qui peuvent être déduites à partir d'un ensemble minimal **F'** appelé **couverture minimale** de F.

Cet ensemble F' est tel que l'ensemble $\mathbf{F^+}$ des dépendances fonctionnelles déduites reste inchangé. Cet ensemble F^+ est appelé **fermeture transitive** de F.

Algorithme de calcul de la couverture minimale :

a) écrire toutes les dépendances fonctionnelles de F sous la forme X -> A, où X est un groupe d'attributs de R, et A un attribut élémentaire. Les dépendances fonctionnelles du type X -> A_1 A_2 … A_n sont remplacées par n dépendances fonctionnelles X -> A_i.

b) essayer de supprimer des dépendances fonctionnelles en utilisant la notion de fermeture d'un ensemble d'attributs, $(X)^+$.

8.4. Décomposition d'un schéma relationnel

La décomposition consiste à remplacer un schéma **R** (A_1, A_2, …, A_n) par un ensemble de schémas **T** = {R_1, R_2, …, R_k} tels que **R** = $R_1 \cup R_2 \cup \ldots \cup R_k$, les schémas R_i n'étant pas obligatoirement disjoints.

Cette opération, qui fait partie du travail de l'administrateur de base de données, a pour objectif d'éliminer les redondances et anomalies de mise à jour évoquées précédemment.

Exemple : Soit le schéma de relation **FOURNISSEUR** (Nom_F, Adresse_F, Produit, Prix).

Les DF s'appliquant sur ce schéma sont les suivantes : **F** = {Nom_F -> Adresse_F ; Nom_F Produit -> Prix}.

Une décomposition possible de ce schéma aboutit aux schémas $\mathbf{R_1}$ = (Nom_F, Adresse_F) et $\mathbf{R_2}$ = (Nom_F, Produit, Prix).

Il est alors possible avec cette décomposition d'ajouter ou de modifier un fournisseur dans la relation $\mathbf{R_1}$ sans incidence sur la relation $\mathbf{R_2}$. Avant de supprimer un fournisseur dans $\mathbf{R_1}$, il faudra quand même s'assurer qu'il n'y a plus de produits correspondant à ce fournisseur dans $\mathbf{R_2}$.

Si **fournisseur** est la relation (table contenant des données) correspondant au schéma **FOURNISSEUR**, alors en décomposant on aura :

$\mathbf{r_1} = \Pi_{\text{Nom\_F Adresse}}$ (**fournisseur**) et $\mathbf{r_2} = \Pi_{\text{Nom\_F Produit Prix}}$ (**fournisseur**)

et la jointure des relations $\mathbf{r_1}$ et $\mathbf{r_2}$ redonnera la relation **founisseur**.

8.5. Décomposition dans perte d'information (SPI)

Soit un schéma de relation **R** décomposé en **T** = {R_1, R_2, …, R_k}, et un ensemble **F** de dépendances fonctionnelles.

La décomposition de **R** est dite SPI "sous **F**" si :

$\forall$ **r** satisfaisant **F**, **r** = Π_{R1} (**r**) jointure Π_{R2} (**r**) … jointure Π_{Rk} (**r**)

La "**méthode des tableaux**" permet de vérifier qu'une décomposition se fait sans perte d'information. Elle comprend deux phases :

a) on construit un tableau avec autant de colonnes que d'attributs A_j, et autant de lignes que de schémas de décomposition R_i,

b) on effectue un remplissage initial du tableau qui correspond à la décomposition envisagée, et on procède de la façon suivante : à l'intersection de la ligne i et de la colonne j, on met $\mathbf{a_j}$ si l'attribut A_j appartient au schéma de relation R_i, $\mathbf{b_{ij}}$ sinon,

c) pour chaque DF X -> Y appartenant à F, on cherche des lignes du tableau pour lesquelles les éléments correspondant à tous les attributs de X sont égaux. Si c'est le cas, on égalise les éléments de ces lignes pour les attributs de Y :

   - si au-moins un des éléments correspondant à Y est égal à $\mathbf{a_j}$, alors tous les autres sont égalisés à $\mathbf{a_j}$,
   - sinon, les éléments sont égalisés à $\mathbf{b_{ij}}$.

En fin de parcours, si une des lignes est remplie de **a**, alors la décomposition est SPI.

Exemple : Soit le schéma de relation **R** (Nom_F, Adresse_F, Produit, Prix), avec comme ensemble de DF **F** = {Nom_F -> Adresse_F ; Nom_F Produit -> Prix}.

Considérons la décomposition de ce schéma en $\mathbf{R_1}$ = (Nom_F, Adresse_F) et $\mathbf{R_2}$ = (Nom_F, Produit, Prix).

Remplissage initial du tableau :

	Nom_F	**Adr_F**	**Produit**	**Prix**
R_1	a_1	a_2		
R_2	a_1		a_3	a_4

Application de la DF Nom_F -> Adr_F :

	Nom_F	**Adr_F**	**Produit**	**Prix**
R_1	a_1	a_2		
R_2	a_1	a_2	a_3	a_4

Exercice : Même question avec :

- **R** (Etudiant, Examen, Heure, Date),
- **F** = {Etudiant Examen -> Heure Date ; Etudiant Heure Date -> Examen}
- $\mathbf{R_1}$ (Etudiant, Examen) et $\mathbf{R_2}$ (Etudiant, Heure, Date).

Exercice : Même question avec :

- **R** (Num_Vol, Date, Porte, Heure, Destination),
- **F** = {Num_Vol Date -> Porte ; Num_Vol -> Destination Heure ; Date Porte Heure -> Num_Vol}
- $\mathbf{R_1}$ (Num_Vol, Destination, Heure) et $\mathbf{R_2}$ (Num_Vol, Porte, Date).

8.6. Décomposition préservant les dépendances (SPD)

Soit un schéma de décomposition **T** = {R_1, R_2, ..., R_k}, et un ensemble **F** de dépendances fonctionnelles.

Soit Π_Z (**F**) l'ensemble des DF {X -> Y} appartenant à $\mathbf{F^+}$ (fermeture de **F**), et tel que XY $\subseteq$ Z.

On dit que **T** préserve les DF si Π_{R1} (**F**) $\cup$ Π_{R2} (**F**) ... $\cup$ Π_{Rk} (**F**) permet de retrouver l'ensemble **F** d'origine.

Remarque : cette contrainte est importante car elle signifie lorsqu'elle est vérifiée que les contraintes d'intégrité de la relation R seront conservées avec la décomposition.

Exemple : Soit le schéma de relation **R** (Nom_F, Adresse_F, Produit, Prix), avec comme ensemble de DF **F** = { Nom_F -> Adresse_F ; Nom_F Produit -> Prix }.

Considérons la décomposition de ce schéma en $\mathbf{R_1}$ = (Nom_F, Adresse_F) et $\mathbf{R_2}$ = (Nom_F, Produit, Prix).

Π_{R1} (**F**) = {Nom_F -> Adresse_F}

Π_{R2} (**F**) = {Nom_F Produit -> Prix}

Π_{R1} (**F**) $\cup$ Π_{R2} (**F**) |= F, et la décomposition préserve donc les dépendances.

Exercice : Même question avec :

- **R** (Etudiant, Examen, Heure, Date),
- **F** = {Etudiant Examen -> Heure Date ; Etudiant Heure Date -> Examen}
- $\mathbf{R_1}$ (Etudiant, Examen) et $\mathbf{R_2}$ (Etudiant, Heure, Date).

Exercice : Même question avec :

- **R** (Num_Vol, Date, Porte, Heure, Destination),
- **F** = {Num_Vol Date -> Porte ; Num_Vol -> Destination Heure ; Date Porte Heure -> Num_Vol}
- $\mathbf{R_1}$ (Num_Vol, Destination, Heure) et $\mathbf{R_2}$ (Num_Vol, Porte, Date).

8.7. 1ère forme normale (FN1)

Définition : une relation est en 1ère forme normale si elle ne contient que des "valeurs atomiques", c'est-à-dire pas de "groupes répétitifs".

Exemple : la valeur "Jacques" affectée à l'attribut Prénom

Contre-exemple : la valeur "Jacques, Paul" affectée à l'attribut Prénom, à moins qu'il ne s'agisse d'un prénom composé.

De la même façon, on ne pourra pas mettre dans un même attribut Parent tous les enfants d'une personne. Il faudra, soit prévoir autant de colonnes que de nombre d'enfants possibles, soit insérer dans la base autant de n-uplets que d'enfants en répétant à chaque fois le nom du parent.

Remarque : cette restriction est très contraignante pour certaines classes d'applications telles que la conception assistée par ordinateur (CAO) ou les systèmes d'information géographique (SIG).

8.8. 2ème forme normale (FN2)

La 2ème forme normale s'appuie sur la notion de DF "**complète**" (ou "pleine").

Une DF X -> Y est complète si, X et Y étant attributs d'une même relation, Y n'est pas fonctionnellement dépendant d'un sous-ensemble de X.

Exemple : si AB -> C sur R (A, B, C), on ne peut avoir ni A -> C ni B -> C.

Dans le cas contraire, c'est-à-dire si on peut enlever un attribut de X, et que la DF est toujours applicable, on dit qu'elle est "**partielle**".

Définition : une relation est en 2ème forme normale si elle est déjà en 1ère forme normale, et si tout attribut n'appartenant pas à la clé (primaire) dépend complètement de cette clé, c'est-à-dire si toutes les DF s'appliquant sur R sont complètes.

Remarque : si toutes les DF sont des DF "simples", avec un seul attribut en partie gauche, ou si les clés sont atomiques, alors la relation est FN2.

8.9. 3ème forme normale (FN3)

En décomposant R en R_1 et R_2, on élimine des risques d'erreurs, mais on peut avoir d'autres types d'erreurs, lors d'opérations de mise à jour, du fait des DF "**transitives**".

Définition : une DF X -> Z est transitive lorsqu'on a X -> Y et Y-> Z (application de la transitivité avec les axiomes d'Armstrong).

Définition : une relation est en troisième forme normale si elle satisfait FN1 et FN2, et si aucun attribut n'appartenant pas à la clé (primaire) ne dépend de la clé

par des DF transitives, c'est-à-dire si aucune DF transitive ne s'applique sur cette relation.

Pour rendre la relation FN3, il faut donc éliminer les DF transitives en plaçant certains attributs dans une autre relation.

Autre définition : Une relation est FN3 si, pour toute DF X -> A s'appliquant sur R avec A non inclus dans X, soit X est clé de R, soit A fait partie d'une clé de R.

Exemple : Soit le schéma de relation **R** (Nom_F, Adresse_F, Produit, Prix), avec comme ensemble de DF **F** = {Nom_F -> Adresse_F ; Nom_F Produit -> Prix}.

La clé de **R** est [Nom_F Produit]. Pour Nom_F -> Adresse_F, Nom_F n'est pas clé, et Adresse_F ne fait pas partie de la clé. La relation n'est donc pas FN3.

8.10. Forme normale de Boyce-Codd (FNBC)

Avec FN3, les DF partielles et transitives ont été éliminées pour les clés primaires, mais il faut également considérer les autres clés possibles (clés "candidates") si elles existent.

Remarque : si la relation ne contient qu'une clé et qu'elle est FN3, alors elle est aussi FNBC.

Définition : une relation est FNBC si elle est FN1, FN2 et FN3, et si toutes les parties gauches des DF sont clés candidates pour la relation.

Autre définition : une relation est FNBC si, pour toute DF X -> A s'appliquant sur R avec A non inclus dans X, X est clé (primaire ou candidate) de R.

Exemple : Soit le schéma de relation **R** (Nom_F, Adresse_F, Produit, Prix), avec comme ensemble de DF **F** = {Nom_F -> Adresse_F ; Nom_F Produit -> Prix}.

La clé de **R** est [Nom_F Produit]. Pour Nom_F -> Adresse_F, Nom_F n'est pas clé, et la relation n'est donc pas FNBC.

8.11. Algorithme de décomposition FNBC (SPI)

Soit un schéma de relation **R**, et une couverture minimale **F'** de DF s'appliquant sur **R**. Une décomposition SPI de **R**, qu'on appellera **D**, est construite de manière itérative :

a) **D** est initialisée à **R**,

b) Soit **T** une relation de **D** qui ne soit pas FNBC. Cela signifie qu'il y a une DF X -> A s'appliquant sur **T** telle que X n'est pas clé de **T**, et que A n'est pas un sous-ensemble de X. On décompose alors **T** en $\mathbf{T_1}$ contenant A et les attributs de X, et $\mathbf{T_2}$ contenant tous les attributs de **T** sauf A.

c) **D** est réinitialisée avec $\mathbf{T_1}$ et $\mathbf{T_2}$, et on boucle sur b) jusqu'à ce que toutes les relations soient FNBC.

8.12. Algorithme de décomposition FN3 (SPI et préservant les DF)

Soit un schéma de relation **R**, et une couverture minimale **F'** de DF s'appliquant sur **R**. Une décomposition SPI et SPD de **R**, qu'on appellera **D**, est construite de la façon suivante :

a) pour chaque DF X -> A, on crée une relation R_i (X, A),

b) si on a plusieurs DF telles que X -> A_1, X -> A_2, ..., X -> A_n, alors on regroupe tous ces attributs dans une même relation R_j (X, A_1, A_2, ..., $A_{n)}$,

c) les attributs n'apparaissant dans aucune relation (pas présents dans les DF) sont regroupés dans une même relation,

d) pour avoir une décomposition SPI, il faut s'assurer qu'il y ait au-moins une clé de R dans une des relations de décomposition. Si ce n'est pas le cas, il faut soit ajouter une relation contenant une clé de R, soit ajouter des attributs dans une des relations de décomposition afin de satisfaire cette contrainte.

9. ORGANISATION ET STOCKAGE DES DONNEES

9.1. Introduction

Organisation physique des données sur disque

Un disque (parfois désigné comme un **volume**) est organisé en **pistes** qui sont elles-mêmes découpées en **secteurs**. Si le disque comporte plusieurs **plateaux**, les pistes de même position sur les différents plateaux constituent un **cylindre**.

L'unité élémentaire d'information qui peut être lue ou écrite sur un disque est le secteur (en général un multiple de 512 octets).

Une opération de lecture ou écriture sur disque se fait en trois temps :

- initialisation de l'opération : sélection de l'unité (ou volume), positionnement des têtes de lecture (piste et secteur),
- opération de transfert proprement dite : lecture ou écriture des données sur disque à une adresse donnée (no. piste / no. cylindre + no. secteur),
- terminaison de l'opération : vérification du bon déroulement de l'opération, signal au système d'exploitation.

Pour diminuer les phases d'initialisation et de terminaison, les données sont lues / écrites de manière temporaire et transparente à l'utilisateur dans des zones mémoire appelées **tampons** ("buffers") qui contiennent plusieurs secteurs. Lorsqu'un tampon est plein / vide, une opération d'écriture / lecture est réalisée.

Schéma Disque

Notions de base de l'organisation physique / logique des données

Un **fichier** est un ensemble d'informations organisées dans des enregistrements logiques / physiques et de longueur fixe ou variable.

Un **enregistrement** est un ensemble d'informations correspondant à une entité logique (par exemple personne) ou physique (bloc ou secteur physique d'un disque). On aura donc des enregistrements logiques (ou articles) en correspondance avec des enregistrements physiques (blocs ou secteurs). Un enregistrement logique pourra être stocké dans un ou plusieurs enregistrements physiques en fonction de sa taille. L'enregistrement logique est l'unité de travail pour l'utilisateur, et l'enregistrement phyique est l'unité de stockage sur disque.

Un **champ** est un sous-ensemble d'un enregistrement qui représente un attribut d'une entité logique (par exemple nom ou adresse d'une personne).

La **clé** est l'identifiant unique d'un enregistrement logique (par exemple no. de sécurité sociale pour une personne).

Schéma Fichier

Types d'organisation physique des données sur disque

On distingue deux types d'organisation des données sur disque

- **l'organisation séquentielle** consiste à placer les enregistrements logiques à la suite les uns des autres dans des enregistrements physiques, indépendamment de leur contenu.
- **l'organisation calculée** elle consiste à placer les enregistrements logiques sur disque en fonction de leur contenu.

L'organisation des données sur disque est fixée lors de la création du fichier. Le placement des données sur disque lors des opérations d'écriture / insertion dépend de ce choix d'organisation.

Cette question sera revue en détail un peu plus loin.

Schéma Organisations

Système de gestion de fichiers (SGF)

Il s'agit d'un des composants du système d'exploitation, dont la tâche est précisément de gérer les fichiers sur disque (création, suprression, ouverture, fermeture) et de prendre en charge les opérations d'accès physique aux fichiers (lecture / écriture) pour des opérations logiques de consultation / insertion / modification / suppression.

Il gère aussi le système de stockage intermédiaire à l'aide de tampons.

Schéma SGF dans SE

9.2. Aspects caractéristiques

Indépendance logique / physique

On a vu au chapitre 1 qu'on distinguait trois niveaux de représentation dans un système d'information : le niveau externe (utilisateurs), le niveau conceptuel (administrateur) et le niveau interne (système de stockage).

L'indépendance logique / physique consiste à libérer l'utilisateur des contraintes d'organisation et d'implantation physiques (structures de données, méthodes d'accès), en ne lui laissant voir (si possible) que l'organisation logique qu'il a lui-même définie.

Performances

L'aspect performances est essentiel pour de nombreuses classes d'applications, et plus particulièrement les **applications transactionnelles** (par exemple systèmes bancaires ou systèmes de réservation, plus généralement tous les **systèmes temps réel**).

Quoiqu'il en soit, le temps de réponse est toujours important pour l'utilisateur, et l'efficacité en ce domaine passe par une organisation physique des données efficace.

Fichiers et structures de données nécessaires au fonctionnement d'un SGBD

Une base de données contient :

a) des fichiers contenant les données entrées par les utilisateurs,

b) un fichier créé par le SGBD, le **dictionnaire des données**, qui contiennent des informations relatives aux données stockées par les utilisateurs (méta-données),

c) des fichiers créés par le SGBD, les **index**, qui permettent d'accélérer les recherches d'informations dans les BD,

d) des données statistiques relatives aux données elles-mêmes (taille des fichiers, nombre de valeurs différentes par attribut, …), mais aussi aux opérations effectuées sur ces données (nombres d'insertions, suppressions et modifications ; fréquences de recherche sur les attributs).

Hiérarchie de mémoires

Un SGBD s'appuie sur différents niveaux de mémoire pour "optimiser" la gestion des données et les opérations relatives aux données :

a) la mémoire principale de l'ordinateur (RAM),

b) la mémoire "**cache**",

c) les disques de stockage,

d) des unités d'archivage tels que des bandes magnétiques ou disques optiques.

Enregistrements de longueur fixe / longueur variable