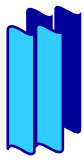

OBJECTIFS ET ARCHITECTURE

DES SGBD

1. INTRODUCTION

Même si vous n'avez jamais utilisé de système de gestion de bases de données (SGBD), vous avez certainement une idée de ce qu'est une base de données (BD) et par là même un SGBD. Une BD est peut-être pour certains une collection de fichiers reliés par des pointeurs multiples, aussi cohérents entre eux que possible, organisés de manière à répondre efficacement à une grande variété de questions. Pour d'autres, une BD peut apparaître comme une collection d'informations modélisant une entreprise du monde réel. Ainsi, un SGBD peut donc être défini comme un ensemble de logiciels systèmes permettant de stocker et d'interroger un ensemble de fichiers interdépendants, mais aussi comme un outil permettant de modéliser et de gérer les données d'une entreprise.

Les données stockées dans des bases de données modélisent des objets du monde réel, ou des associations entre objets. Les objets sont en général représentés par des articles de fichiers, alors que les associations correspondent naturellement à des liens entre articles. Les données peuvent donc être vues comme un ensemble de fichiers reliés par des pointeurs ; elles sont interrogées et mises à jour par des programmes d'applications écrits par les utilisateurs ou par des programmes utilitaires fournis avec le SGBD (logiciels d'interrogation interactifs, éditeurs de rapports, etc.). Les programmes sont écrits dans un langage de programmation traditionnel appelé langage de 3^{e} génération (C, COBOL, FORTRAN, etc.) ou dans un langage plus avancé intégrant des facilités de gestion d'écrans et d'édition de rapports appelé langage de 4^{e} génération (Visual BASIC, SQL/FORMS, MANTIS, etc.). Dans tous les cas, ils

accèdent à la base à l'aide d'un langage unifié de description et manipulation de données permettant les recherches et les mises à jour (par exemple, le langage SQL). Cette vision simplifiée d'un SGBD et de son environnement est représentée figure II.1.

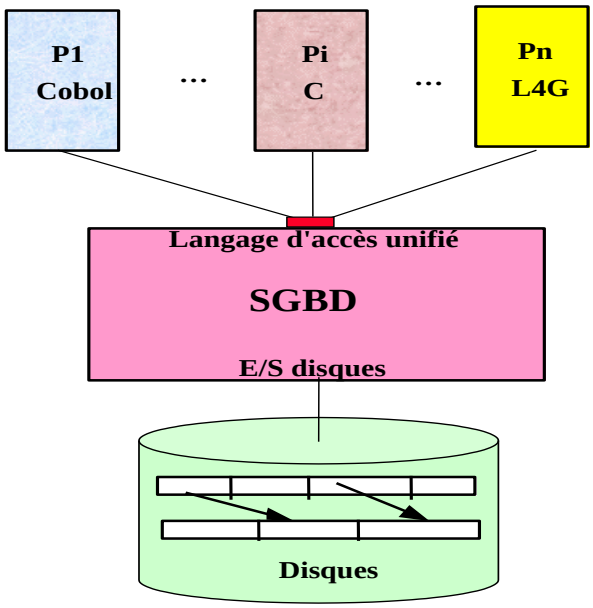

Figure II.1 — Composants d'un environnement base de données

L'objectif de ce chapitre est d'essayer de clarifier la notion plus ou moins floue de SGBD. Pour cela nous présentons d'abord les objectifs que ces systèmes cherchent à atteindre. Bien sûr, peu de SGBD satisfont pleinement tous ces objectifs, mais ils y tendent tous plus ou moins. Ensuite, nous exposerons les méthodes et concepts de base nécessaires à la compréhension générale du fonctionnement des SGBD, puis l'architecture fonctionnelle de référence proposée par le groupe de normalisation ANSI/X3/SPARC. Une bonne compréhension de cette architecture est essentielle à la compréhension des SGBD proposés jusqu'à ce jour. Pour conclure le chapitre, nous étudierons diverses architectures opérationnelles proposées par des groupes de normalisation ou des constructeurs de SGBD, telles l'architecture client-serveur à deux ou trois states (*2-tiers* ou *3-tiers*) implantée aujourd'hui par beaucoup de constructeurs.

2. MODÉLISATION DES DONNÉES

Une idée centrale des bases de données est de séparer la description des données effectuée par les administrateurs de la manipulation effectuée par les programmes d'application. La description permet de spécifier les structures et les types de données de l'application alors que la manipulation consiste à effectuer des interrogations, des insertions et des mises à jour. Dès 1965, l'idée de décrire les données des applications de manière indépendante des traitements fut proposée. Aujourd'hui, plusieurs niveaux de description gérés par un SGBD permettent de réaliser des abstractions progressives des données stockées sur disques, de façon à s'approcher de la vision particulière de chaque utilisateur.

2.1 Instances et schémas

Toute description de données consiste à définir les propriétés d'ensembles d'objets modélisés dans la base de données, et non pas d'objets particuliers. Les objets particuliers sont définis par les programmes d'applications lors des insertions et mises à jour de données. Ils doivent alors vérifier les propriétés des ensembles auxquels ils appartiennent. On distingue ainsi deux notions essentielles :

- le **type d'objet** permet de spécifier les propriétés communes à un ensemble d'objets en termes de structures de données visible et d'opérations d'accès,
- l'**instance d'objet** correspond à un objet particulier identifiable parmi les objets d'un type.

Bien qu'**occurrence** soit aussi employé, on préfère aujourd'hui le terme d'**instance**. Nous précisons ci-dessous ces notions en les illustrant par des exemples.

> **Notion II.1 : Type d'objet (*Object type*)**
>
> Ensemble d'objets possédant des caractéristiques similaires et manipulables par des opérations identiques.

Exemples :

1. Le type d'objet Entier = { 0, ± 1,… ± N,… ± ∞ } muni des opérations standards de l'arithmétique { +, *, / , - } est un type d'objet élémentaire, supporté par tous les systèmes.
2. Le type d'objet Vin possédant les propriétés Cru, Millésime, Qualité, Quantité peut être muni des opérations Produire et Boire, qui permettent respectivement d'accroître et de décroître la quantité. C'est un type d'objet composé pouvant être utilisé dans une application particulière, gérant par exemple des coopératives vinicoles.
3. Le type d'objet Entité possédant les propriétés P1, P2,…Pn et muni des opérations Créer, Consulter, Modifier, Détruire est un type d'objet générique qui permet de modéliser de manière très générale la plupart des objets du monde réel.

> **Notion II.2 : Instance d'objet (*Object instance*)**
>
> Elément particulier d'un type d'objets, caractérisé par un identifiant et des valeurs de propriétés.

Exemples :

1. L'entier 10 est une instance du type Entier.

2. Le vin (Volnay, 1992, Excellent, 1000) est une instance du type Vin.
3. L'entité e(a1, a2, …an), où a1, a2,…,an désignent des valeurs particulières, est une instance du type Entité.

Toute description de données s'effectue donc au niveau du type, à l'aide d'un ensemble d'éléments descriptifs permettant d'exprimer les propriétés d'ensembles d'objets et composant un **modèle de description de données**. Ce dernier est souvent représenté par un formalisme graphique. Il est mis en œuvre à l'aide d'un **langage de description de données (LDD)**. La description d'un ensemble de données particulier, correspondant par exemple à une application, à l'aide d'un langage de description, donne naissance à un **schéma de données.** On distingue généralement le schéma source spécifié par les **administrateurs de données** et le schéma objet résultant de la compilation du précédent par une machine. Le schéma objet est directement utilisable par le système de gestion de bases de données afin de retrouver et de vérifier les propriétés des instances d'objets manipulées par les programmes d'applications.

> **Notion II.3 : Modèle de description de données (*Data model*)**
>
> Ensemble de concepts et de règles de composition de ces concepts permettant de décrire des données.

> **Notion II.4 : Langage de description de données (*Data description language*)**
>
> Langage supportant un modèle et permettant de décrire les données d'une base d'une manière assimilable par une machine.

> **Notion II.5 : Schéma (*Schema*)**
>
> Description au moyen d'un langage déterminé d'un ensemble de données particulier.

2.2 Niveaux d'abstraction

Un objectif majeur des SGBD est d'assurer une abstraction des données stockées sur disques pour simplifier la vision des utilisateurs. Pour cela, trois niveaux de description de données ont été distingués par le groupe ANSI/X3/SPARC [ANSI78, Tsichritzis78]. Ces niveaux ne sont pas clairement distingués par tous les SGBD : ils sont mélangés en deux niveaux dans beaucoup de systèmes existants. Cependant, la conception d'une base de données nécessite de considérer et de spécifier ces trois niveaux, certains pouvant être pris en compte par les outils de génie logiciel aidant à la construction des applications autour du SGBD.

2.2.1 Le niveau conceptuel

Le niveau central est le niveau conceptuel. Il correspond à la structure canonique des données qui existent dans l'entreprise, c'est-à-dire leur structure sémantique inhérente sans souci

d'implantation en machine, représentant la vue intégrée de tous les utilisateurs. La définition du **schéma conceptuel** d'une entreprise ou d'une application n'est pas un travail évident. Ceci nécessite un accord sur les concepts de base que modélisent les données. Par exemple, le schéma conceptuel permettra de définir :

1. les types de données élémentaires qui définissent les propriétés élémentaires des objets de l'entreprise (cru d'un vin, millésime, qualité, etc.) ;

2. les types de données composés qui permettent de regrouper les attributs afin de décrire les objets du monde réel ou les relations entre objets (vin, personne, buveur, etc.) ;

3. les types de données composés qui permettent de regrouper les attributs afin de décrire les associations du monde réel (abus de vin par un buveur, production d'un vin par un producteur, etc.) ;

4. éventuellement, des règles que devront suivre les données au cours de leur vie dans l'entreprise (l'âge d'une personne est compris entre 0 et 150, tout vin doit avoir un producteur, etc.).

Un exemple de schéma conceptuel défini en termes de types d'objets composés (souvent appelés entités) et d'associations entre ces objets est représenté figure II.2. Le type d'objet Buveur spécifie les propriétés d'un ensemble de personnes qui consomment des vins. Le type d'objet Vin a été introduit ci-dessous. Une consommation de vin (abusivement appelée ABUS) associe un vin et un buveur. La consommation s'effectue à une date donnée en une quantité précisée.

Type d'objets :
BUVEUR(Nom, Prénom, Adresse)
VIN(Cru, Millésime, Qualité, Quantité)
Type d'associations :
ABUS(BUVEUR, VIN, Date, Quantité)

Figure II.2 — Exemple de schéma conceptuel

2.2.2 Le niveau interne

Le niveau interne correspond à la structure de stockage supportant les données. La définition du **schéma interne** nécessite au préalable le choix d'un SGBD. Elle permet donc de décrire les données telles qu'elles sont stockées dans la machine, par exemple :

- les fichiers qui les contiennent (nom, organisation, localisation,…) ;

- les articles de ces fichiers (longueur, champs composants, modes de placement en fichiers,…) ;

- les chemins d'accès à ces articles (index, chaînages, fichiers inversés,…).

Une implémentation possible des données représentées figure II.2 est illustrée figure II.3. Il existe un fichier indexé sur le couple (Cru, Millésime) dont chaque article contient successivement le cru, le millésime, la qualité, la quantité stockée de vin. Il existe un autre fichier décrivant les buveurs et leurs abus. Chaque article de ce deuxième fichier contient le nom, le prénom et l'adresse d'un buveur suivi d'un groupe répétitif correspondant aux abus comprenant le nombre d'abus et pour chacun d'eux un pointeur sur le vin bu, la date et la quantité. Un index sur le nom du buveur permet d'accéder directement aux articles de ce fichier. Un autre index permet aussi d'y accéder par la date des abus.

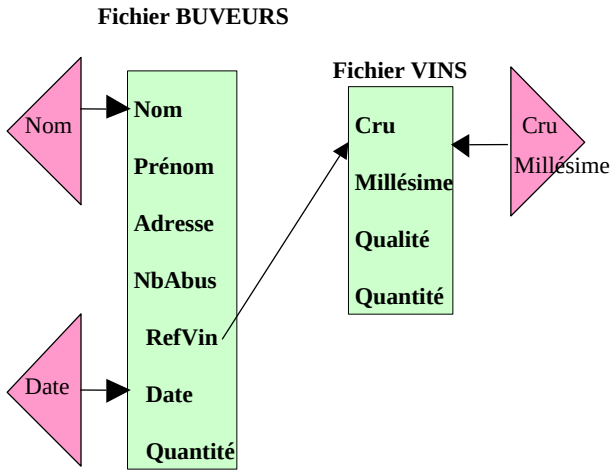

Figure II.3 — Exemple de schéma interne

2.2.3 Le niveau externe

Au niveau externe, chaque groupe de travail utilisant des données possède une description des données perçues, appelée **schéma externe**. Cette description est effectuée selon la manière dont le groupe voit la base dans ses programmes d'application. Alors qu'au niveau conceptuel et interne les schémas décrivent toute une base de données, au niveau externe ils décrivent simplement la partie des données présentant un intérêt pour un utilisateur ou un groupe d'utilisateurs. En conséquence, un schéma externe est souvent qualifié de vue externe. Le modèle externe utilisé est dépendant du langage de manipulation de la base de données utilisé. La figure II.4 donne deux exemples de schéma externe pour la base de données dont le schéma conceptuel est représenté figure II.2. Il est à souligner que la notion de schéma externe permet d'assurer une certaine sécurité des données. Un groupe de travail ne peut en effet accéder qu'aux données décrites dans son schéma externe. Les autres données sont ainsi protégées contre les accès non autorisés ou mal intentionnés de la part de ce groupe de travail.

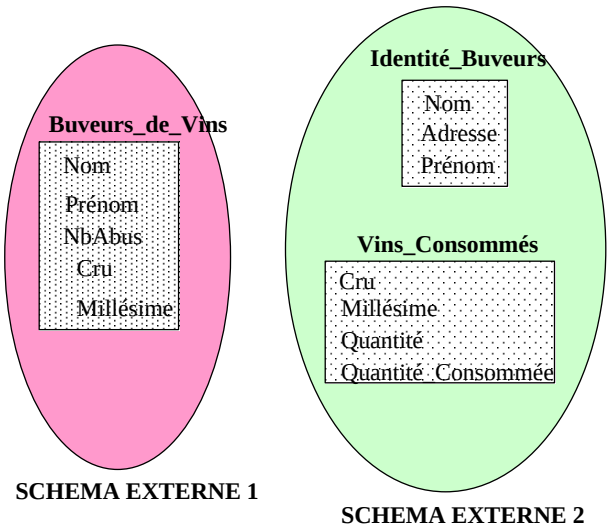

Figure II.4 — Exemples de schémas externes

2.2.4 Synthèse des niveaux de schémas

Retenez que, pour une base particulière, il existe un seul schéma interne et un seul schéma conceptuel. En revanche, il existe en général plusieurs schémas externes. Un schéma externe peut être défini par un groupe d'utilisateurs. A partir de là, il est possible de construire des schémas externes pour des sous-groupes du groupe d'utilisateurs considéré. Ainsi, certains schémas externes peuvent être déduits les uns des autres. La figure II.5 illustre les différents schémas d'une base de données centralisée. Nous rappelons ci-dessous ce que représente chacun des niveaux de schéma à l'aide d'une notion.

Notion II.6 : Schéma conceptuel (*Conceptual Schema*)

Description des données d'une entreprise ou d'une partie d'une entreprise en termes de types d'objets et de liens logiques indépendants de toute représentation en machine, correspondant à une vue canonique globale de la portion d'entreprise modélisée.

Notion II.7 : Schéma interne (*Internal Schema*)

Description des données d'une base en termes de représentation physique en machine, correspondant à une spécification des structures de mémorisation et des méthodes de stockage et d'accès utilisées pour ranger et retrouver les données sur disques.

Notion II.8 : Schéma externe (*External Schema*)

Description d'une partie de la base de données extraite ou calculée à partir de la base physique, correspondant à la vision d'un programme ou d'un utilisateur, donc à un arrangement particulier de certaines données.

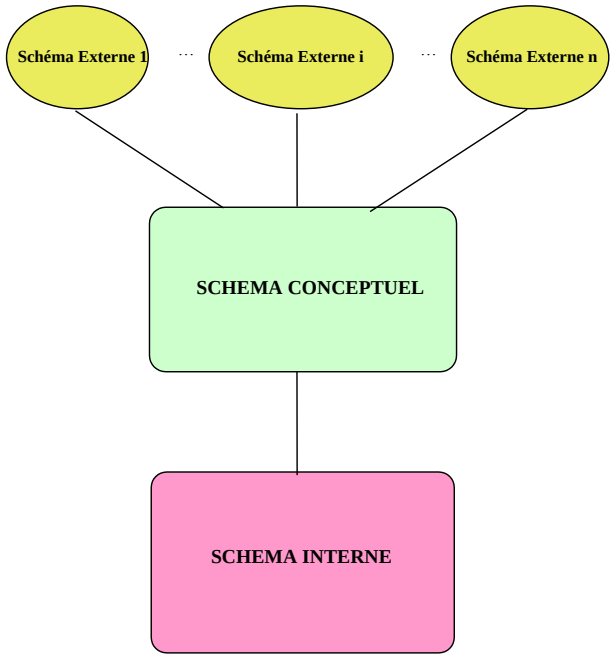

Figure II.5 — Les trois niveaux de schémas

2.3 Le modèle entité-association

Les données élémentaires décrivent des événements atomiques du monde réel. Par exemple, la donnée « 10 000 francs » peut correspondre à une instance de salaire ou de prix. Dupont Jules est un nom. Dans les bases de données, des instances de types élémentaires sont groupées ensemble pour constituer un objet composé. L'abstraction qui concatène des données élémentaires (et plus généralement des objets) est appelée **l'agrégation**. La figure II.6 représente par un graphe l'agrégation des données (Volnay, 1978, Excellente, 100) pour constituer un objet composé décrivant le vin identifié par Volnay78.

Notion II.9 : Agrégation (*Aggregation*)

Abstraction consistant à grouper des objets pour constituer des objets composés d'une concaténation d'objets composants.

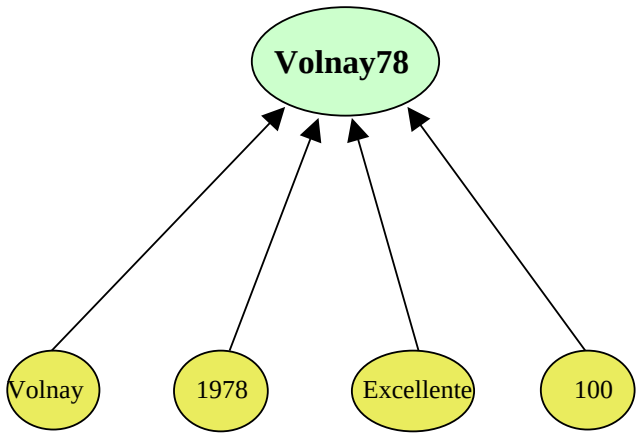

Figure II. 6 — Exemple d'agrégation de valeurs

Le modèle entité-association [Chen76] est basé sur une perception du monde réel qui consiste à distinguer des agrégations de données élémentaires appelées **entités** et des liaisons entre entités appelées **associations**. Intuitivement, une entité correspond à un objet du monde réel généralement défini par un nom, par exemple un vin, un buveur, une voiture, une commande, etc. Une entité est une agrégation de données élémentaires. Un type d'entité définit un ensemble d'entités constitué par des données de même type. Les types de données agrégées sont appelés les attributs de l'entité ; ils définissent ses propriétés.

Notion II.10 : Entité (*Entity*)

Modèle d'objet identifié du monde réel dont le type est défini par un nom et une liste de propriétés.

Une association correspond à un lien logique entre deux entités ou plus. Elle est souvent définie par un verbe du langage naturel. Une association peut avoir des propriétés particulières définies par des attributs spécifiques.

Notion II.11 : Association (*Relationship*)

Lien logique entre entités dont le type est défini par un verbe et une liste éventuelle de propriétés.

Notion II.12 : Attribut (*Attribute*)

Propriété d'une entité ou d'une association caractérisée par un nom et un type élémentaire.

Le modèle entité-association, qui se résume aux trois concepts précédents, permet de modéliser simplement des situations décrites en langage naturel : les noms correspondent aux entités, les verbes aux associations et les adjectifs ou compléments aux propriétés. Il s'agit là bien sûr d'une abstraction très schématique d'un sous-ensemble réduit du langage naturel que nous illustrons par un exemple simple.

Exemple :

Les buveurs abusent de vins en certaines quantités à des dates données. Tout buveur a un nom, un prénom, une adresse et un type. Un vin est caractérisé par un cru, un millésime, une qualité, une quantité et un degré.

Il est possible de se mettre d'accord au niveau conceptuel pour représenter une telle situation par le schéma entité-association suivant :

- entités = {BUVEUR, VIN};
- association = {ABUS};
- attributs = {Nom, Prénom, Adresse et Type pour BUVEUR; Cru, Millésime, Qualité, Quantité et Degré pour VIN; Quantité et Date pour ABUS}.

Un des mérites essentiels du modèle entité-association est de permettre une représentation graphique élégante des schémas de bases de données [Chen76]. Un rectangle représente une entité ; un losange représente une association entre entités ; une ellipse représente un attribut. Les losanges sont connectés aux entités qu'ils associent par des lignes. Les attributs sont aussi connectés aux losanges ou rectangles qu'ils caractérisent. La figure II.7 représente le diagramme entité-association correspondant à la situation décrite dans l'exemple ci-dessus.

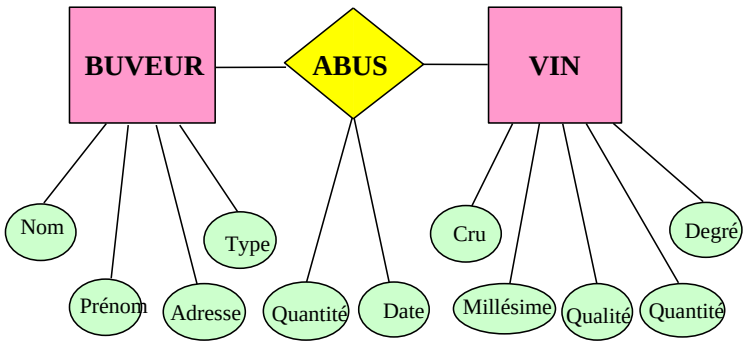

Figure II.7 — Exemple de diagramme entité-association

3. OBJECTIFS DES SGBD

Le principal objectif d'un SGBD est d'assurer l'indépendance des programmes aux données, c'est-à-dire la possibilité de modifier les schémas conceptuel et interne des données sans modifier les programmes d'applications, et donc les schémas externes vus par ces programmes. Cet objectif est justifié afin d'éviter une maintenance coûteuse des programmes lors des modifications des structures logiques (le découpage en champs et articles) et physiques (le mode de stockage) des données. Plus précisément, on distingue l'indépendance physique qui permet de changer les schémas internes sans changer les programmes d'applications, et l'indépendance logique qui permet de modifier les schémas conceptuels (par exemple, ajouter un type d'objet) sans changer les programmes d'applications.

Afin d'assurer encore une meilleure indépendance des programmes aux données est rapidement apparue la nécessité de manipuler (c'est-à-dire d'interroger et de mettre à jour) les données par des langages de haut niveau spécifiant celles que l'on veut traiter (le quoi) et non pas comment y accéder. Ainsi, les procédures d'accès aux données restent invisibles aux programmes d'application qui utilisent donc des langages non procéduraux. Ces langages référencent des descriptions logiques des données (les schémas externes) stockées dans le dictionnaire de données. Les descriptions de données, qui existent à plusieurs niveaux introduits ci-dessus, sont établies par les administrateurs de données. Un SGBD se doit donc de faciliter l'administration (c'est-à-dire la création et la modification de la description) des données. En résumé, voici les objectifs premiers d'un SGBD :

- Indépendance physique des programmes aux données
- Indépendance logique des programmes aux données
- Manipulation des données par des langages non procéduraux
- Administration facilitée des données.

Les SGBD conduisent à mettre en commun les données d'une entreprise, ou au moins d'une application dans une base de données décrite par un dictionnaire de données. Cette mise en commun ne va pas sans problèmes d'efficacité: de nombreux utilisateurs accèdent simultanément aux données souvent situées sur un même disque. La base de données devient ainsi un goulot d'étranglement. Il faut assurer globalement l'efficacité des accès. Il faut aussi garantir les utilisateurs contre les mises à jour concurrentes, et donc assurer le partage des données. L'environnement multi-usager nécessite de protéger la base de données contre les mises à jour erronées ou non autorisées: il faut assurer la cohérence des données. Notamment, des données redondantes doivent rester égales. Enfin, en cas de panne système, ou plus simplement d'erreurs de programmes, il faut assurer la sécurité des données, en permettant par exemple de repartir sur des versions correctes. En résumé, voici les objectifs additionnels des SGBD, qui sont en fait des conséquences des objectifs premiers :

- Efficacité des accès aux données

- Partage des données
- Cohérence des données
- Redondance contrôlée des données
- Sécurité des données.

Dans la pratique, ces objectifs ne sont que très partiellement atteints. Ci-dessous nous analysons plus précisément chacun d'eux.

3.1 Indépendance physique

Bien souvent, les données élémentaires du monde réel sont assemblées pour décrire les objets et les associations entre objets directement perceptibles dans le monde réel. Bien que souvent deux groupes de travail assemblent différemment des données élémentaires, il est possible au sein d'une entreprise bien organisée de définir une structure canonique des données, c'est-à-dire un partitionnement en ensembles et sous-ensembles ayant des propriétés bien définies et cohérentes avec les vues particulières. Cet assemblage peut être considéré comme l'intégration des vues particulières de chaque groupe de travail. Il obéit à des règles qui traduisent l'essentiel des propriétés des données élémentaires dans le monde réel. Il correspond au schéma conceptuel d'une base de données.

Par opposition, la structure de stockage des données appartient au monde des informaticiens et n'a donc un sens que dans l'univers du système informatique. Le schéma interne décrit un assemblage physique des données en articles, fichiers, chemins d'accès (organisation et méthode d'accès des fichiers, modes de placement des articles dans les fichiers, critères de tri, chaînages…) sur des mémoires secondaires. Cet assemblage propre au monde informatique doit être basé sur des considérations de performances et de souplesse d'accès.

Un des objectifs essentiels des SGBD est donc de permettre de réaliser l'indépendance des structures de stockage aux structures de données du monde réel [Stonebraker74], c'est-à-dire entre le schéma interne et le schéma conceptuel. Bien sûr, ces deux schémas décrivent les mêmes données, mais à des niveaux différents. Il s'agit donc de pouvoir modifier le schéma interne sans avoir à modifier le schéma conceptuel, en tenant compte seulement des critères de performance et de flexibilité d'accès. On pourra par exemple ajouter un index, regrouper deux fichiers en un, changer l'ordre ou le codage des données dans un article, sans mettre en cause les entités et associations définies au niveau conceptuel.

Les avantages de l'indépendance physique peuvent être facilement compris si l'on considère les inconvénients de la non-indépendance physique. Celle-ci impliquerait que la manière dont les données sont organisées sur mémoire secondaire soit directement l'image de l'organisation canonique de données dans le monde réel. Pour permettre de conserver les possibilités d'optimisation de performances vitales aux systèmes informatiques, les notions de méthodes d'accès, modes de placement, critères de tri, chaînages et codages de données devraient directement apparaître dans le monde réel et donc dans les applications. Tout changement informatique devrait alors être répercuté dans la vie d'une entreprise et par

conséquent impliquerait une reconstruction des applications. Cela est bien sûr impraticable, d'où la nécessité d'indépendance des structures de stockages aux données du monde réel.

3.2 Indépendance logique

Nous avons admis ci-dessus l'existence d'un schéma conceptuel modélisant les objets et associations entre objets dans le monde réel. Ce schéma résulte d'une synthèse des vues particulières de chaque groupe de travail utilisant la base de données, c'est-à-dire d'une intégration de schémas externes. En conséquence, chaque groupe de travail réalisant une application doit pouvoir assembler différemment les données pour former par exemple les entités et les associations de son schéma externe, ou plus simplement des tables qu'il souhaite visualiser. Ainsi, chacun doit pouvoir se concentrer sur les éléments constituant son centre d'intérêt, c'est-à-dire qu'un utilisateur doit pouvoir ne connaître qu'une partie des données de la base au travers de son schéma externe, encore appelé **vue**.

Il est donc souhaitable de permettre une certaine indépendance des données vues par les applications à la structure canonique des données de l'entreprise décrite dans le schéma conceptuel. L'indépendance logique est donc la possibilité de modifier un schéma externe sans modifier le schéma conceptuel. Elle assure aussi l'indépendance entre les différents utilisateurs, chacun percevant une partie de la base via son schéma externe, selon une structuration voire un modèle particulier.

Les avantages de l'indépendance logique [Date71] sont les suivants :

- permettre à chaque groupe de travail de voir les données comme il le souhaite ;
- permettre l'évolution de la vue d'un groupe de travail (d'un schéma externe) sans remettre en cause, au moins dans une certaine mesure, le schéma conceptuel de l'entreprise ;
- permettre l'évolution d'un schéma externe sans remettre en cause les autres schémas externes.

En résumé, il doit être possible d'ajouter des attributs, d'en supprimer d'autres, d'ajouter et de supprimer des associations, d'ajouter ou de supprimer des entités, etc., dans des schémas externes mais aussi dans le schéma conceptuel sans modifier la plus grande partie des applications.

3.3 Manipulation des données par des langages non procéduraux

Les utilisateurs, parfois non professionnels de l'informatique, doivent pouvoir manipuler simplement les données, c'est-à-dire les interroger et les mettre à jour sans préciser les algorithmes d'accès. Plus généralement, si les objectifs d'indépendance sont atteints, les utilisateurs voient les données indépendamment de leur implantation en machine. De ce fait, ils doivent pouvoir manipuler les données au moyen de langages non procéduraux, c'est-à-dire en décrivant les données qu'ils souhaitent retrouver (ou mettre à jour) sans décrire la manière de les retrouver (ou de les mettre à jour) qui est propre à la machine. Les langages

non procéduraux sont basés sur des assertions de logique du premier ordre. Ils permettent de définir les objets désirés au moyen de relations entre objets et de propriétés de ces objets.

Deux sortes d'utilisateurs manipulent en fait les bases de données : les utilisateurs interactifs et les programmeurs. Les utilisateurs interactifs peuvent interroger voire mettre à jour la base de données. Ils sont parfois non informaticiens et réclament des langages simples. De tels langages peuvent être formels (logique du premier ordre) ou informels (menus). Une large variété de langages interactifs doivent être supportés par un SGBD, depuis les langages de commandes semi-formels jusqu'aux langages graphiques, en passant par l'interrogation par menus ou par formes. La limite supérieure de tels langages est le langage naturel, qui reste cependant en général trop complexe et lourd pour interroger les bases de données.

Les programmeurs écrivent des programmes en utilisant des langages traditionnels dits de 3^{e} génération (C, COBOL, PL1, etc.), des langages plus récents orientés objet tels C++ ou Java ou des langages de 4^{e} génération (VB, PL/SQL, FORTÉ, etc.). Ces derniers regroupent des instructions de programmation structurée (WHILE, IF, CASE, etc.), des expressions arithmétiques, des commandes d'accès à la base de données et des commandes d'édition et entrée de messages (menus déroulants, gestion de fenêtres, rapports imprimés, etc.). Ils sont de plus en plus souvent orientés objets. Dans tous les cas, il est important que le SGBD fournisse les commandes nécessaires de recherche et mise à jour de données pour pouvoir accéder aux bases. Une intégration harmonieuse avec le langage de programmation, qui traite en général un objet à la fois, est souhaitable.

3.4 Administration facilitée des données

Un SGBD doit fournir des outils pour décrire les données, à la fois leurs structures de stockage et leurs présentations externes. Il doit permettre le suivi de l'adéquation de ces structures aux besoins des applications et autoriser leur évolution aisée. Les fonctions qui permettent de définir les données et de changer leur définition sont appelées outils d'administration des données. Afin de permettre un contrôle efficace des données, de résoudre les conflits entre divers points de vue pas toujours cohérents, de pouvoir optimiser les accès aux données et l'utilisation des moyens informatiques, on a pensé à centraliser ces fonctions entre les mains d'un petit groupe de personnes hautement qualifiées, appelées administrateurs de données.

En fait, la centralisation des descriptions de données entre les mains d'un groupe spécialisé a souvent conduit à des difficultés d'organisation. Aussi, l'évolution des SGBD modernes tend à fournir des outils permettant de décentraliser la description de données, tout en assurant une cohérence entre les diverses descriptions partielles. Un dictionnaire de données dynamique pourra ainsi aider les concepteurs de bases de données. Pour permettre une évolution rapide, les descriptions de données devront être faciles à consulter et à modifier. L'évolution va donc vers le développement d'outils intégrés capables de faciliter l'administration des données et d'assurer la cohérence des descriptions.

3.5 Efficacité des accès aux données

Les performances en termes de débit (nombre de transactions types exécutées par seconde) et de temps de réponse (temps d'attente moyen pour une requête type) sont un problème clé des SGBD. L'objectif de débit élevé nécessite un *overhead* minimal dans la gestion des tâches accomplie par le système. L'objectif de bons temps de réponse implique qu'une requête courte d'un utilisateur n'attende pas une requête longue d'un autre utilisateur. Il faut donc partager les ressources (unités centrales, unités d'entrées-sorties) entre les utilisateurs en optimisant l'utilisation globale et en évitant les pertes en commutation de contextes.

Le goulot d'étranglement essentiel dans les systèmes de bases de données reste les E/S disques. Une E/S disque coûte en effet quelques dizaines de millisecondes. Afin de les éviter, on utilisera une gestion de tampons en mémoire centrale dans de véritables mémoires caches des disques, afin qu'un grand nombre d'accès aux données se fasse en mémoire. Un autre facteur limitatif est dû à l'utilisation de langages non procéduraux très puissants afin d'interroger et mettre à jour la base de données. Ainsi, il devient possible de demander en une requête le tri d'un grand volume de données. Il devient donc aussi nécessaire d'optimiser l'activité de l'unité centrale pour traiter les opérations en mémoire. En résumé, un SGBD devra chercher à optimiser une fonction de coût de la forme C(Q) = a * ES(Q) + b * UC(Q) pour un ensemble typique de requêtes (recherches et mises à jour) Q ; ES(Q) est le nombre d'entrées-sorties réalisées pour la requête Q et UC(Q) est le temps unité centrale dépensé ; a et b sont des facteurs convertissant entrées-sorties et temps d'unité centrale en coûts.

3.6 Redondance contrôlée des données

Dans les systèmes classiques à fichiers non intégrés, chaque application possède ses données propres. Cela conduit généralement à de nombreuses duplications de données avec, outre la perte en mémoire secondaire associée, un gâchis important en moyens humains pour saisir et maintenir à jour plusieurs fois les mêmes données. Avec une approche base de données, les fichiers plus ou moins redondants seront intégrés en un seul fichier partagé par les diverses applications. L'administration centralisée des données conduisait donc naturellement à la non-duplication physique des données afin d'éviter les mises à jour multiples.

En fait, avec les bases de données réparties sur plusieurs calculateurs interconnectés, il est apparu souhaitable de faire gérer par le système des copies multiples de données. Cela optimise les performances en interrogation, en évitant les transferts sur le réseau et en permettant le parallélisme des accès. On considère donc aujourd'hui que la redondance gérée par le SGBD au niveau physique des données n'est pas forcément mauvaise. Il faudra par contre éviter la redondance anarchique, non connue du système, qui conduirait les programmes utilisateurs à devoir mettre à jour plusieurs fois une même donnée. Il s'agit donc de bien contrôler la redondance, qui permet d'optimiser les performances, en la gérant de manière invisible pour les utilisateurs.

3.7 Cohérence des données

Bien que les redondances anarchiques entre données soient évitées par l'objectif précédent, les données vues par l'utilisateur ne sont pas indépendantes. Au niveau d'ensemble de

données, il peut exister certaines dépendances entre données. Par exemple, une donnée représentant le nombre de commandes d'un client doit correspondre au nombre de commandes dans la base. Plus simplement, une donnée élémentaire doit respecter un format et ne peut souvent prendre une valeur quelconque. Par exemple, un salaire mensuel doit être supérieur à 4 700 F et doit raisonnablement rester inférieur à 700 000 F. Un système de gestion de bases de données doit veiller à ce que les applications respectent ces règles lors des modifications des données et ainsi assurer la cohérence des données. Les règles que doivent explicitement ou implicitement suivre les données au cours de leur évolution sont appelées contraintes d'intégrité.

3.8 Partage des données

L'objectif est ici de permettre aux applications de partager les données de la base dans le temps mais aussi simultanément. Une application doit pouvoir accéder aux données comme si elle était seule à les utiliser, sans attendre mais aussi sans savoir qu'une autre application peut les modifier concurremment.

En pratique, un utilisateur exécute des programmes généralement courts qui mettent à jour et consultent la base de données. Un tel programme interactif, appelé transaction, correspond par exemple à l'entrée d'un produit en stock ou à une réservation de place d'avion. Il est important que deux transactions concurrentes (par exemple, deux réservations sur le même avion) ne s'emmêlent pas dans leurs accès à la base de données (par exemple, réservent le même siège pour deux passagers différents). On cherchera donc à assurer que le résultat d'une exécution simultanée de transactions reste le même que celui d'une exécution séquentielle dans un ordre quelconque des transactions.

3.9 Sécurité des données

Cet objectif a deux aspects. Tout d'abord, les données doivent être protégées contre les accès non autorisés ou mal intentionnés. Il doit exister des mécanismes adéquats pour autoriser, contrôler ou enlever les droits d'accès de n'importe quel usager à tout ensemble de données. Les droits d'accès peuvent également dépendre de la valeur des données ou des accès précédemment effectués par l'usager. Par exemple, un employé pourra connaître les salaires des personnes qu'il dirige mais pas des autres employés de l'entreprise.

D'un autre côté, la sécurité des données doit aussi être assurée en cas de panne d'un programme ou du système, voire de la machine. Un bon SGBD doit être capable de restaurer des données cohérentes après une panne disque, bien sûr à partir de sauvegardes. Aussi, si une transaction commence une mise à jour (par exemple un transfert depuis votre compte en banque sur celui de l'auteur) et est interrompue par une panne en cours de mise à jour (par exemple après avoir débité votre compte en banque), le SGBD doit assurer l'intégrité de la base (c'est-à-dire que la somme d'argent gérée doit rester constante) et par suite défaire la transaction qui a échoué. Une transaction doit donc être totalement exécutée, ou pas du tout : il faut assurer l'atomicité des transactions, et ainsi garantir l'intégrité physique de la base de données.

4. FONCTIONS DES SGBD

Cette section présente les fonctions essentielles d'un SGBD. Un SGBD permet de décrire les données des bases, de les interroger, de les mettre à jour, de transformer des représentations de données, d'assurer les contrôles d'intégrité, de concurrence et de sécurité. Il supporte de plus en plus fréquemment des fonctions avancées pour la gestion de procédures et d'événements. Toutes ces fonctionnalités sont illustrées par des exemples simples.

4.1 Description des données

Un SGBD offre donc des interfaces pour décrire les données. La définition des différents schémas est effectuée par les **administrateurs** de données ou par les personnes jouant le rôle d'administrateur.

> **Notion II.13 : Administrateur de données (*Data Administrator*)**
>
> Personne responsable de la définition de schémas de bases de données.

Dans un SGBD ou un environnement de développement de bases de données supportant trois niveaux de schémas, les administrateurs de données ont trois rôles :

- **Administrateur de bases de données**. L'exécutant de ce rôle est chargé de la définition du schéma interne et des règles de correspondance entre les schémas interne à conceptuel.
- **Administrateur d'entreprise**. Le porteur de ce rôle est chargé de la définition du schéma conceptuel.
- **Administrateur d'application.** L'attributaire est chargé de la définition des schémas externes et des règles de correspondance entre les schémas externe et conceptuel.

Ces trois rôles peuvent être accomplis par les mêmes personnes ou par des personnes différentes. Un rôle essentiel est celui d'administrateur d'entreprise, qui inclut la définition des informations que contient la base de données au niveau sémantique, par exemple avec des diagrammes entité-association. La plupart des SGBD modernes supportent seulement un schéma interne et plusieurs schémas externes. Le schéma conceptuel est défini en utilisant un outil d'aide à la conception (par exemple au sein d'un atelier de génie logiciel) s'appuyant généralement sur des interfaces graphiques permettant d'élaborer des diagrammes de type entité-association.

Quoi qu'il en soit, les différents schémas et procédures pour passer de l'un à l'autre sont stockés dans le **dictionnaire des données**. Celui-ci peut être divisé en deux dictionnaires : le dictionnaire d'entreprise qui contient le schéma conceptuel et les procédures et commentaires s'appliquant sur ce schéma, et le dictionnaire des bases qui contient les schémas internes et externes, ainsi que les procédures de passage d'un niveau à l'autre. Tout dictionnaire contient en général des descriptions en langage naturel permettant de préciser la signification des données. Un dictionnaire de données peut contenir des informations non strictement bases de données, telles que des masques d'écrans ou des programmes. Les informations sont souvent

stockées en format source, mais aussi en format compilé. Un dictionnaire de données organisé sous forme de base de données est appelé **métabase**.

> **Notion II.14 : Dictionnaire des données (*Data Dictionnary*)**
>
> Ensemble des schémas et des règles de passage entre les schémas associés à une base de données, combinés à une description de la signification des données.

> **Notion II.15 : Métabase (*Metabase*)**
>
> Dictionnaire de données organisé sous forme de base de données qui décrit donc les autres bases.

Un SGBD fournit des commandes permettant de définir les schémas interne, conceptuel et externe. Afin d'illustrer concrètement cette fonctionnalité, voici les commandes essentielles permettant de créer un schéma avec le seul modèle présenté jusque-là, c'est-à-dire le modèle entité-association. La syntaxe dérive d'une extension du langage QUEL [Zook77] au modèle entité-association.

Voici donc les commandes minimales nécessaires :

- pour créer une base de données :

```
CREATDB <nom-de-base>
```

- pour créer une entité :

```
CREATE ENTITY <nom-d'entité> (<nom-d'attribut> <type>
[{,<nom-d'attribut> <type>}…])
```

- pour créer une association :

```
CREATE RELATIONSHIP <nom-d'association> (<nom-d'entité>
[{,<nom d'entité>}…], [{,<nom-d'attribut> <type>}…])
```

- pour détruire une entité ou une association :

```
DESTROY {<nom-de-relation> | <nom-d'association>}
```

- pour détruire une base :

```
DESTROYDB <nom-de-base>.
```

Ces commandes permettent de créer un schéma conceptuel entité-association. Elles sont utilisées dans la figure II.8 pour créer la base de données correspondant au schéma conceptuel de la figure II.7.

```
CREATE ENTITY Buveur
(Nom Char(16), Prénom Char(16), Adresse Text, Type Char(4));
CREATE ENTITY Vin
(Cru Char(10), Millésime Int, Qualité Char(10), Quantité Int, Degré Real)
CREATE RELATIONSHIP Abus
(Buveur, Vin, Date Date, Quantité Int)
```

Figure II.8 — Exemple de description de données

D'autres commandes sont nécessaires, par exemple pour créer le schéma interne. Un exemple typique à ce niveau est une commande de création d'index sur un ou plusieurs attributs d'une entité :

```
CREATE INDEX ON <nom-d'entité>
USING <nom-d'attribut> [{,<nom-d'attribut>}…]
```

Nous verrons plus loin des commandes de création de vues (schémas externes).

4.2 Recherche de données

Tout SGBD fournit des commandes de recherche de données. Les SGBD modernes offrent un langage d'interrogation assertionnel permettant de retrouver les données par le contenu sans préciser la procédure d'accès. Les SGBD de première génération offraient des langages procéduraux permettant de rechercher un objet dans la base de données par déplacements successifs. Afin d'illustrer un langage de requête non procédural, nous introduisons informellement un langage dérivé du langage QUEL adapté au modèle entité-association.

Le langage QUEL [Zook77] est le langage de manipulation de données du système INGRES [Stonebraker76], un des premiers systèmes relationnels développé à l'université de Californie Berkeley et commercialisé sur de nombreuses machines. Ce langage est aujourd'hui peu utilisé car il a été remplacé par SQL, langage plus commercial. Il a cependant le mérite d'être à la fois simple et didactique, car dérivé de la logique du premier ordre. Nous proposons une variante simplifiée étendue au modèle entité-association [Zaniolo83].

Afin d'exprimer des questions, QUEL permet tout d'abord la définition de variables représentant un objet quelconque d'une entité ou d'une association. Une définition de variables s'effectue à l'aide de la clause :

```
RANGE OF variable IS {nom-d'entité | nom d'association}.
```

La variable est associée avec l'entité ou l'association spécifiée. Plusieurs variables associées à plusieurs entités ou associations peuvent être déclarées par une clause RANGE. Les variables déclarées demeurent en principe connues jusqu'à une nouvelle déclaration ou la fin de session.

Dans le cas de la base de données créée figure II.8, on peut par exemple définir trois variables :

RANGE OF B IS Buveur;

RANGE OF V IS Vin;

RANGE OF A IS Abus.

Une commande de recherche permet de retrouver les données de la base répondant à un critère plus ou moins complexe, appelé **qualification**. Une qualification est une expression logique (ET de OU par exemple) de critères simples, chaque critère permettant soit de comparer un attribut à une valeur, soit de parcourir une association. En QUEL, un attribut est spécifié par X.Attribut, où X est une variable et Attribut un nom d'attribut de l'entité ou de l'association représentée par la variable. Un critère simple sera donc de la forme X.Attribut = valeur pour une recherche sur valeur, ou X.Entité = Y pour un parcours d'association. D'autres fonctionnalités sont possibles, comme nous le verrons plus loin dans cet ouvrage. Une qualification est une expression logique de critères simples.

> **Notion II.16 : Qualification de question (*Query Qualification*)**
>
> Expression logique construite avec des OU (OR), ET (AND), NON (NOT) de critères simples permettant d'exprimer une condition que doit satisfaire les résultats d'une question.

Une recherche s'exprime alors à l'aide de requête du type suivant, où la liste résultat est une suite d'attributs de variables ou de fonctions appliquées à ces attributs :

```
RETRIEVE (liste résultat)
[WHERE qualification] ;
```

Voici quelques questions simples afin d'illustrer les capacités minimales d'un SGBD.

(Q1) Rechercher les noms et adresses des buveurs :

```
RETRIEVE B.Nom, B.Adresse;
```

(Q2) Rechercher les crus et millésimes des vins de qualité excellente :

```
RETRIEVE V.Cru, V.Millésime
WHERE V.Qualité = "Excellente" ;
```

(Q3) Rechercher les noms des gros buveurs ainsi que les crus, dates et quantités de vins qu'ils ont consommé :

```
RETRIEVE B.Nom, V.Cru, A.Date, A.Quantité
WHERE B.Type = "Gros" AND A.Buveur = B AND A.Vin = V ;
```

A priori, un SGBD doit offrir un **langage complet**, c'est-à-dire un langage permettant de poser toutes les questions possibles sur la base de données. On limite cependant aux langages du premier ordre la notion de complétude. Les questions peuvent faire intervenir un grand nombre d'entités et d'associations, des calculs, des quantificateurs (quel que soit, il existe), des restructurations de données, etc. En restant au premier ordre, il n'est pas possible de quantifier des ensembles d'objets.

> **Notion II.17 : Langage complet (*Complete Language*)**
>
> Langage de requêtes permettant d'exprimer toutes les questions que l'on peut poser en logique du premier ordre à une base de données.

4.3 Mise à jour des données

Le concept de mise à jour intègre à la fois l'insertion de données dans la base, la modification de données et la suppression de données. Nous illustrons ces aspects par une variation du langage QUEL adapté au modèle entité association.

Le langage présenté comporte une commande pour insérer des instances dans la base dont la syntaxe est la suivante :

```
APPEND [TO] <nom-d'entité> [(<liste-d'attributs>)]
(<liste-de-valeurs>) [{,(<liste-de-valeurs>)}]… ;
```

Cette commande permet d'ajouter les instances définies par les listes de valeurs à l'entité de nom <nom-d'entité>. Plusieurs instances d'entités peuvent ainsi être ajoutées dans la base. La liste d'attributs permet de spécifier les seuls attributs que l'on désire documenter, les autres étant a priori remplacés par une valeur nulle signifiant inconnue. Par exemple, l'ajout du vin <Volnay, 1978, Excellente, 100> dans la base s'effectuera par la commande :

```
(U1)    APPEND TO Vin (Cru, Millésime, Qualité, Quantité)
        (Volnay, 1978, Excellente, 100) ;
```

L'insertion d'instances d'association est plus délicate car il faut insérer un enregistrement référençant des entités de la base. A titre indicatif, cela peut être fait par une commande APPEND TO dans laquelle les références aux entités connectées sont des variables calculées par une qualification. On aboutit alors à une insertion qualifiée du type :

```
APPEND [TO] <nom-d'association> [(<liste-d'attributs>)]
<liste-de-valeurs> [{,<liste-de-valeurs>}]…
WHERE <qualification> ;
```

Une liste de valeurs peut alors comprendre des attributs extraits de la base par la qualification. Par exemple, la commande suivante insère un abus de Volnay 78 au buveur Dupont :

```
(U2)    APPEND TO abus(buveur, vin, date, quantité)
```

```
        (V, B, 10-02-92, 100)
        WHERE B.Nom = "Dupont" AND V.Cru = "Volnay"
        AND V. Millésime = 1978 ;
```

La modification de données s'effectue en général par recherche des données à modifier à l'aide d'une qualification, puis par renvoi dans la base des données modifiées. La commande peut être du style suivant :

```
REPLACE <variable><attribut> = <valeur>
        [{,<attribut> = <valeur>}…]
[WHERE qualification]
```

Elle permet de changer la valeur des attributs figurant dans la liste pour tous les tuples de la variable satisfaisant la qualification. Par exemple, l'ajout de 1 000 litres aux stocks de Volnay s'effectuera par la commande (V est supposée une variable sur l'entité Vin) :

```
(U3)    REPLACE V (Quantité = Quantité + 1.000)
        WHERE V.Cru = "Volnay" ;
```

Finalement, il est aussi possible de supprimer des tuples d'une base de données par la commande très simple :

```
DELETE variable
WHERE <qualification>
```

Par exemple, la suppression de tous les vins de millésime 1992 s'effectue par :

```
(U4)    DELETE V
        WHERE V.Millésime = 1992 ;
```

4.4 Transformation des données

Comme il peut exister plusieurs niveaux de schémas gérés par système pour décrire un même ensemble de données, un système de gestion de base de données doit pouvoir assurer le passage des données depuis le format correspondant à un niveau dans le format correspondant à un autre niveau. Cette fonction est appelée **transformation de données**.

> **Notion II.18 : Transformation de données (*Data mapping*)**
>
> Fonction effectuant la restructuration d'instances de données conformes à un schéma en instances de données conformes à un autre schéma.

Dans un SGBD à trois niveaux de schémas, il existera donc deux niveaux de transformation :

- la transformation conceptuelle - interne permettant de faire passer des instances de données depuis le format conceptuel au format interne et réciproquement ;

- la transformation externe - conceptuelle permettant de faire passer des instances de données depuis le format conceptuel au format externe et réciproquement.

A titre d'exemple, la figure II.9 représente la transformation d'un ensemble d'occurrences de données depuis le format conceptuel indiqué au format externe indiqué.

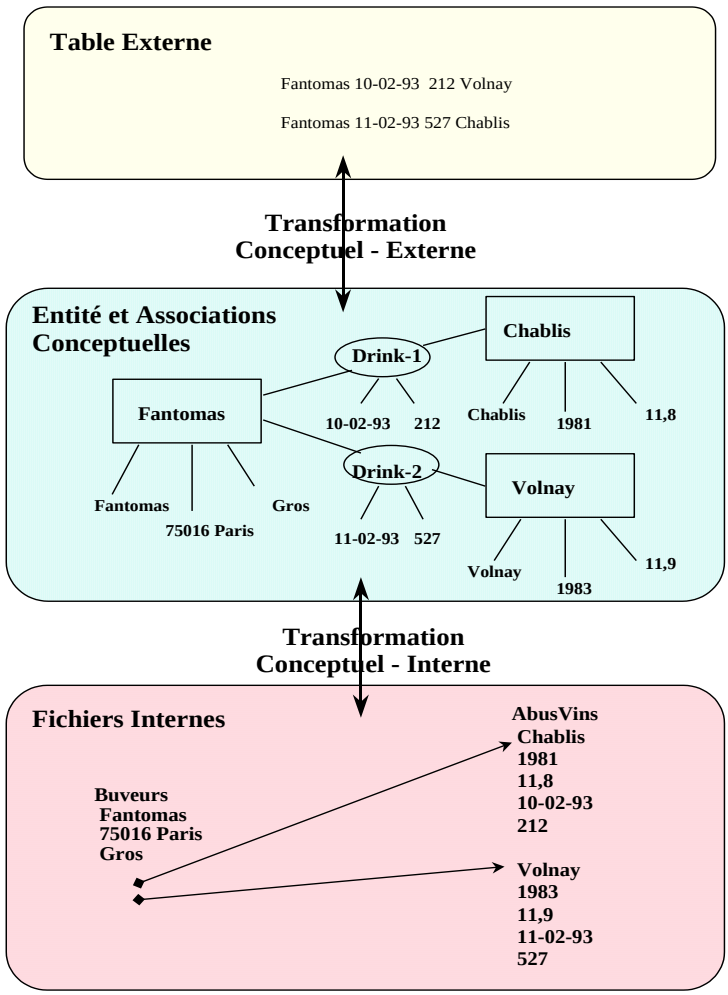

Figure II.9 — Transformation de données

Pour être capable d'effectuer automatiquement la transformation des données d'un niveau à un autre, un SGBD doit connaître les correspondances existant entre les niveaux. Pour cela, lors de la définition des schémas, le groupe d'administration des données doit expliciter comment les schémas se déduisent les uns des autres au moyen de règles de correspondance. Ces règles sont souvent exprimer sous la forme de questions.

Notion II.19 : Règles de correspondance (*Mapping rules*)

Questions définissant les procédures de transformation des données depuis un niveau de schéma dans un autre niveau.

Dans les systèmes de gestion de base de données classiques, les règles de correspondance sont bien souvent mélangées avec les schémas. Il y a cependant intérêt à distinguer ces deux notions. Par exemple, le langage QUEL permet de définir une vue de la base (schéma externe) par une commande du type suivant :

```
DEFINE VIEW <nom d'entité> (<liste-d'attributs>) AS
RETRIEVE (liste résultat)
[WHERE qualification] ;
```

La règle de correspondance entre l'entité de la vue (une table en QUEL) et le schéma de la base est clairement exprimée par une question. On pourra par exemple définir le schéma externe Gros_Buveurs comme suit, B désignant une variable sur Buveur :

```
(V1)    DEFINE VIEW Gros_Buveurs (Nom, Prénom, Adresse) AS
        RETRIEVE B.Nom, B.Prénom, B.Adresse
        WHERE B.Type = "Gros";
```

4.5 Contrôle de l'intégrité des données

Comme on l'a vu au niveau des objectifs, un SGBD doit assurer le maintien de la cohérence des données par rapport aux schémas (contrôles de type), mais aussi entre elles (contrôle de redondance). On appelle **contrainte d'intégrité** toute règle implicite ou explicite que doivent suivre les données. Par exemple, si le SGBD supporte un modèle entité-association, les contraintes suivantes sont possibles :

1. Toute entité doit posséder un identifiant unique attribué par l'utilisateur. Pour les vins, nous avons supposé jusque-là que cru et millésime constituaient un identifiant. Il pourra être plus sage de numéroter les vins par un attribut numéro de vin (noté NV). Cet attribut devra être un identifiant unique, donc toujours documenté (non nul). Une telle contrainte est souvent appelé **contrainte d'unicité de clé.**

2. Certaines associations doivent associer des instances d'entité obligatoirement décrites dans la base. Ainsi, un abus ne peut être enregistré que pour un buveur et un vin existants dans la base. Une telle contrainte est souvent appelée **contrainte référentielle.**

3. Tout attribut d'entité ou d'association doit posséder une valeur qui appartient à son type. Par exemple, une quantité doit être un nombre entier. Il est même possible de préciser le domaine de variation permis pour un attribut ; par exemple, une quantité de

vin peut varier entre 0 et 10 000. Une telle contrainte est souvent appelée **contrainte de domaine**.

> **Notion II. 20 : Contrainte d'intégrité (*Integrity Constraint*)**
>
> Règle spécifiant les valeurs permises pour certaines données, éventuellement en fonction d'autres données, et permettant d'assurer une certaine cohérence de la base de données.

En résumé, un grand nombre de type de contraintes d'intégrité est possible. Celles-ci gagnent à être déclarées au SGBD par une commande spécifique DEFINE INTEGRITY. Le SGBD doit alors les vérifier lors des mises à jour de la base.

4.6 Gestion de transactions et sécurité

La gestion de transactions permet d'assurer qu'un groupe de mises à jour est totalement exécuté ou pas du tout. Cette propriété est connue sous le nom d'**atomicité** des transactions. Elle est garantie par le SGBD qui connaît l'existence de transactions à l'aide de deux commandes : BEGIN_TRANSACTION et END_TRANSACTION. Ces commandes permettent d'assurer que toutes les mises à jour qu'elles encadrent sont exécutées ou qu'aucune ne l'est.

> **Notion II.21 : Atomicité des transactions (*Transaction Atomicity*)**
>
> Propriété d'une transaction consistant à être totalement exécutée ou pas du tout.

Une transaction est donc un groupe de mises à jour qui fait passer la base d'un état à un autre état. Les états successifs doivent être cohérents et donc respecter les contraintes d'intégrité. Cette responsabilité incombe au programmeur qui code la transaction. Cette propriété est connue sous le nom de **correction des transactions**.

> **Notion II.22 : Correction des transactions (*Transaction Correctness*)**
>
> Propriété d'une transaction consistant à respecter la cohérence de la base de données en fin d'exécution.

Lorsqu'une transaction est partiellement exécutée, les données peuvent passer par des états incohérents transitoires, qui seront corrigés par les mises à jour suivantes de la transaction. Pendant cette période d'activité, les effets de la transaction ne doivent pas être visibles aux autres transactions. Cette propriété est connue sous le nom d'**isolation des transactions** ; l'isolation doit être assurée par le SGBD.

> **Notion II.23 : Isolation des transactions (*Transaction Isolation*)**
>
> Propriété d'une transaction consistant à ne pas laisser visible à l'extérieur les données modifiées avant la fin de la transaction.

En résumé, un bon SGBD doit donc assurer les trois propriétés précédentes pour les transactions qu'il gère : Atomicité, Correction, Isolation. Ces propriétés sont parfois résumées

par le sigle ACID, le D signifiant que l'on doit aussi pouvoir conserver durablement les mises à jour des transactions (en anglais, *durability*). En plus, le SGBD doit garantir la sécurité des données. Rappelons que la sécurité permet d'éviter les accès non autorisés aux données par des mécanismes de contrôle de droits d'accès, mais aussi de restaurer des données correctes en cas de pannes ou d'erreurs.

4.7 Autres fonctions

De nombreuses autres fonctions se sont progressivement intégrées aux SGBD. Par exemple, beaucoup savent aujourd'hui déclencher des procédures cataloguées par l'utilisateur lors de l'apparition de certaines conditions sur les données ou lors de l'exécution de certaines opérations sur certaines entités ou associations. Cette fonctionnalité est connue sous le nom de **déclencheur**, encore appelé réflexe dans le contexte des architectures client-serveur en relationnel. Les déclencheurs permettent de rendre les bases de données actives, par exemple en déclenchant des procédures de correction lors de l'apparition de certains événements. Il s'agit là d'une fonctionnalité nouvelle qui prend de plus en plus d'importance.

> **Notion II.24 : Déclencheur (*Trigger*)**
>
> Mécanisme permettant d'activer une procédure cataloguée lors de l'apparition de conditions particulières dans la base de données.

De manière plus générale, les SGBD sont amenés à supporter des règles permettant d'inférer (c'est-à-dire de calculer par des raisonnements logiques) de nouvelles données à partir des données de la base, lors des mises à jour ou des interrogations. Cela conduit à la notion de SGBD déductif, capable de déduire des informations à partir de celles connues et de règles de déduction.

Enfin, les SGBD sont aussi amenés à gérer des objets complexes, tels des dessins d'architecture ou des cartes de géographie, en capturant finement le découpage de ces gros objets en sous-objets composants. Ces objets pourront être atteints via des procédures elles-mêmes intégrées au SGBD. Cela conduit à la notion de SGBD à objets, capable de gérer des objets multiples manipulés par des fonctions utilisateurs.

5. ARCHITECTURE FONCTIONNELLE DES SGBD

5.1 L'architecture à trois niveaux de l'ANSI/X3/SPARC

Les groupes de normalisation se sont penchés depuis fort longtemps sur les architectures de SGBD. A la fin des années 70, le groupe ANSI/X3/SPARC DBTG a proposé une architecture intégrant les trois niveaux de schémas : externe, conceptuel et interne. Bien qu'ancienne [ANSI78], cette architecture permet de bien comprendre les niveaux de description et transformation de données possible dans un SGBD.

L'architecture est articulée autour du dictionnaire de données et comporte deux parties :

1. un ensemble de modules (appelés processeurs) permettant d'assurer la description de données et donc la constitution du dictionnaire de données ;

2. une partie permettant d'assurer la manipulation des données, c'est-à-dire l'interrogation et la mise à jour des bases.

Dans chacune des parties, on retrouve les trois niveaux interne, conceptuel et externe. L'architecture proposée est représentée figure II.10.

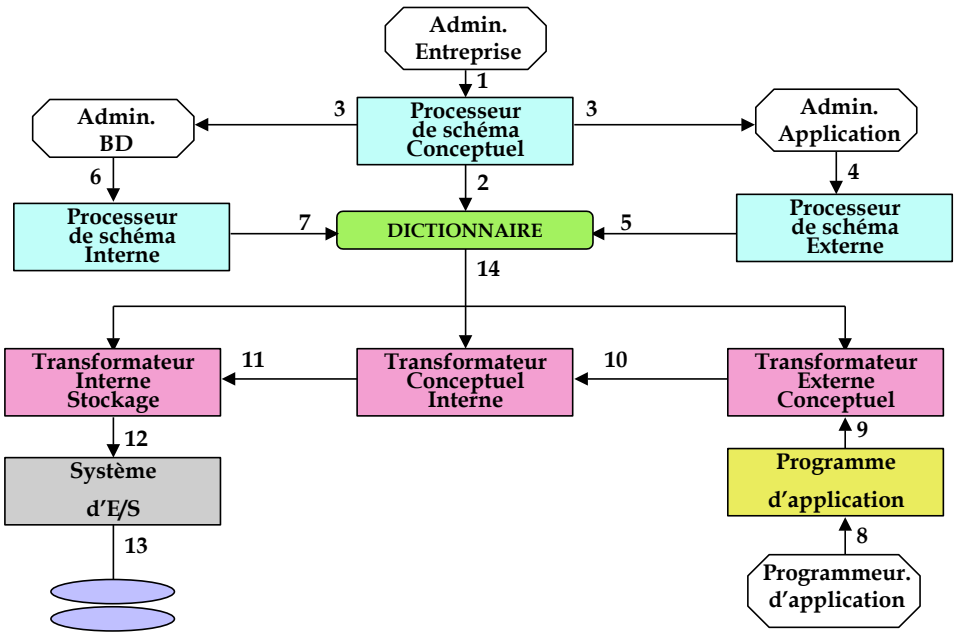

Figure II.10 — L'Architecture ANSI/X3/SPARC

Les fonctions de chacun des processeurs indiqués sont les suivantes. Le processeur de schéma conceptuel compile le schéma conceptuel et, dans le cas où il n'y a pas d'erreur, range ce schéma compilé dans le dictionnaire des données. Le processeur de schéma externe compile les schémas externes et les règles de correspondance externe à conceptuel et, après une compilation sans erreur, range le schéma compilé et les règles de correspondance dans le dictionnaire des données. Le processeur de schéma interne a un rôle symétrique pour le schéma interne.

Le processeur de transformation externe à conceptuel traduit les manipulations externes en manipulations conceptuelles et dans l'autre sens les données conceptuelles en données externes. Une requête externe peut donner naissance à plusieurs requêtes au niveau conceptuel. Le processeur de transformation conceptuel à interne traduit les manipulations conceptuelles en manipulations internes et dans l'autre sens les données internes en données conceptuelles. Finalement, le processeur de transformation interne à stockage traduit les manipulations internes en programmes d'accès au système de stockage et délivre les données stockées en format correspondant au schéma interne.

Les diverses interfaces indiquées correspondent successivement à (les numéros se rapportent à la figure II.10) :

(1) Langage de description de données conceptuel, format source ; il permet à l'administrateur d'entreprise de définir le schéma conceptuel en format source. Il correspond par exemple aux commandes CREATE ENTITY et CREATE RELATIONSHIP vues ci-dessus (paragraphe II.4.1).

(2) Langage de description de données conceptuel, format objet ; il résulte de la compilation du précédent et permet de ranger le schéma objet dans le dictionnaire des données.

(3) Description de données conceptuel, format d'édition ; cette interface permet aux administrateurs d'applications et de bases de consulter le schéma conceptuel afin de définir les règles de correspondance. Il pourrait s'agir par exemple d'une visualisation graphique des diagrammes entité-association.

(4) Langages de description de données externes, format source ; ils peuvent être multiples si le SGBD supporte plusieurs modèles de données. Ils permettent aux administrateurs d'applications de définir les schémas externes et les règles de correspondance avec le schéma conceptuel. Par exemple, la commande DEFINE VIEW introduite ci-dessus illustre ce type de langage, qui permet donc de définir des schémas externes encore appelés vues.

(5) Langages de description de données externes, format objet ; ils correspondent à la compilation des précédents et permettent de ranger les schémas externes objets dans le dictionnaire de données.

(6) Langages de description de données internes, format source ; il permet à l'administrateur de bases de données de définir le schéma interne et les données de correspondance avec le schéma conceptuel. Par exemple, la commande CREATE INDEX vue ci-dessus (paragraphe II.4.1) se situe à ce niveau d'interface.

(7) Langage de description de données internes, format objet ; il correspond à la compilation du précédent et permet de ranger le schéma interne objet dans le dictionnaire des données.

(8) Langages de manipulation de données externes, format source ; ils permettent aux programmeurs d'applications, voire aux non-informaticiens, de manipuler les données décrites dans un schéma externe. Ils comportent des commandes du type RETRIEVE, APPEND, MODIFY et DELETE référençant les objets décrits dans un schéma externe.

(9) Langages de manipulation de données externes, format objet ; ils correspondent aux schémas compilés des précédents.

(10) Langage de manipulation de données conceptuelle, format objet ; il est généré par les processeurs de transformation externe à conceptuel afin de manipuler les données logiques décrites dans le schéma conceptuel. Ils comportent des primitives

correspondant à la compilation des commandes du type RETRIEVE, APPEND, MODIFY et DELETE référençant cette fois les objets décrits dans le schéma conceptuel.

(11) Langage de manipulation de données interne, format objet ; il est généré par le processeur de transformation conceptuel à interne afin de manipuler les données internes. Il permet par exemple d'accéder aux articles de fichiers via des index.

(12) Langage de stockage de données, format objet ; il correspond à l'interface du système de stockage de données. Il permet par exemple de lire ou d'écrire une page dans un fichier.

(13) Interface mémoires secondaires ; elle permet d'effectuer les entrées-sorties sur les disques.

(14) Interface d'accès au dictionnaire des données ; elle permet aux divers processeurs de transformation d'accéder aux schémas objets et aux règles de correspondance.

5.2 Une architecture fonctionnelle de référence

L'architecture à trois niveaux de schémas présentée ci-dessus permet de bien comprendre les niveaux de description et de manipulation de données. Cependant, elle n'est que peu suivie, la plupart des systèmes intégrant le niveau interne et le niveau conceptuel dans un seul niveau. En fait, le véritable niveau conceptuel est pris en charge par les outils d'aide à la conception à l'extérieur du SGBD, lors de la conception de l'application. Nous proposons une architecture de référence (voir figure II.11) plus proche de celles des SGBD actuels, basée sur seulement deux niveaux de schéma : le schéma et les vues. Le schéma correspond à une intégration des schémas interne et conceptuel, une vue est un schéma externe.

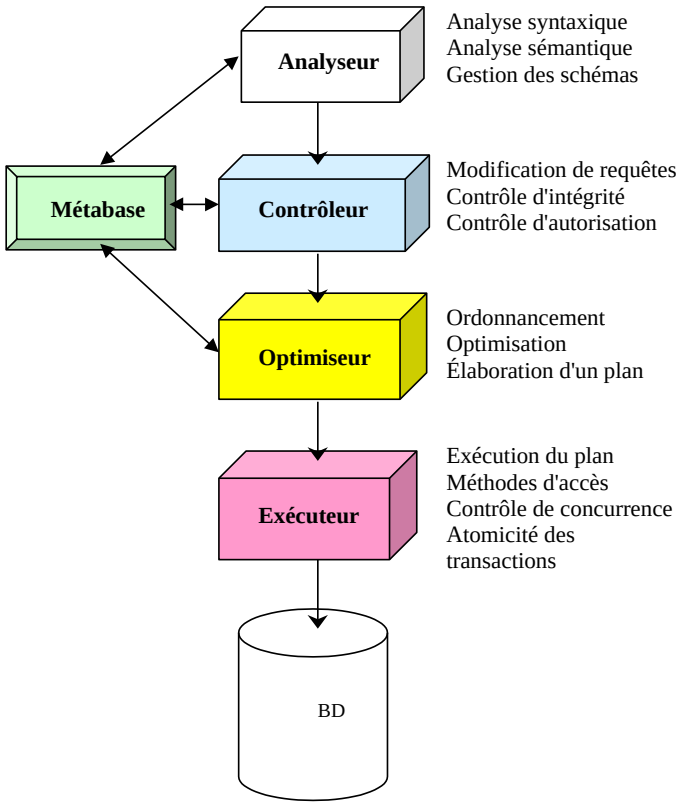

Figure II.11 — Architecture typique d'un SGBD

Du point de vue de la description de données, un SGBD gère un dictionnaire de données, encore appelé **métabase** car souvent organisé comme une base de données qui décrit les autres bases. Ce dictionnaire est alimenté par les commandes de définition du schéma (par exemple CREATE ENTITY, CREATE RELATIONSHIP, CREATE INDEX) et de définition des vues (par exemple DEFINE VIEW). Ces commandes sont analysées et traitées par le processeur d'analyse (ANALYSEUR), plus spécifiquement par la partie traitant le langage de description de données de ce processeur. Celle-ci fait souvent appel aux fonctions plus internes du SGBD pour gérer le dictionnaire comme une véritable base de données.

Du point de vue de la manipulation des données, les requêtes (par exemple, RETRIEVE, APPEND, MODIFY, DELETE) sont tout d'abord prises en compte par l'**analyseur de requêtes**. Celui-ci réalise l'analyse syntaxique (conformité à la grammaire) et sémantique (conformité à la vue référencée ou au schéma) de la requête. Celle-ci est alors traduite en format interne, les noms étant remplacés par des références internes.

Une requête en format interne référençant une vue doit tout d'abord être traduite en une (ou plusieurs) requête(s) référençant des objets existant dans la base, c'est-à-dire des objets décrits au niveau du schéma. Cette fonctionnalité, accomplie au niveau du **contrôleur de requêtes** figure II.11, est souvent appelée **modification de requêtes**, car elle consiste à changer la requête en remplaçant les références aux objets de la vue par leur définition en termes d'objets du schéma. C'est aussi au niveau du contrôleur que sont pris en compte les problèmes de contrôle de droits d'accès (autorisation de lire ou d'écrire un objet) et de contrôle d'intégrité lors des mises à jour. Le contrôle d'intégrité consiste à vérifier que la base

n'est pas polluée lors des mises à jour, c'est-à-dire que les règles de cohérence des données restent vérifiées après mise à jour.

L'**optimiseur de requêtes** est un composant clé du SGBD. Son rôle essentiel est d'élaborer un plan d'accès optimisé pour traiter la requête. Pour se faire, il décompose en général la requête en opérations d'accès élémentaires (e.g., sélection d'index, lecture d'article, etc.) et choisit un ordre d'exécution optimal ou proche de l'optimum pour ces opérations. Il choisit aussi les méthodes d'accès à utiliser. Pour effectuer les meilleurs choix, l'optimiseur s'appuie souvent sur un modèle de coût qui permet d'évaluer le coût d'un plan d'accès avant son exécution. Le résultat de l'optimisation (le plan d'accès optimisé) peut être sauvegardé en mémoire pour des exécutions multiples ultérieures ou exécuté directement puis détruit.

L'**exécuteur de plans** a enfin pour rôle d'exécuter le plan d'accès choisi et élaboré par l'optimiseur. Pour cela, il s'appuie sur les méthodes d'accès qui permettent d'accéder aux fichiers via des index et/ou des liens. C'est aussi à ce niveau que sont gérés les problèmes de concurrence d'accès et d'atomicité de transactions. Les techniques utilisées dépendent beaucoup de l'architecture opérationnelle du SGBD qui s'exprime en termes de processus et de tâches.

5.3 L'architecture du DBTG CODASYL

Le groupe de travail *Data Base Task Group* du comité CODASYL responsable du développement de COBOL a proposé depuis 1971 des recommandations pour construire un système de bases de données [Codasyl71]. Ces recommandations comportent essentiellement des langages de description de données et de manipulation de données orientés COBOL que nous étudierons plus loin, mais aussi une recommandation d'architecture.

L'architecture d'un système obéissant aux recommandations CODASYL s'articule autour du schéma qui permet de définir les articles, leurs données et les liens entre ces articles. Ce schéma peut être assimilé au schéma conceptuel de l'architecture ANSI/X3/SPARC, bien que comportant, à notre avis, pas mal de notions du niveau interne. La structure de stockage des données est définie par le schéma de stockage qui correspond plus ou moins au schéma interne ANSI. La notion de schéma externe est définie pour COBOL et est appelée sous-schéma. Un groupe de travail a également proposé des langages de description de sous-schémas pour FORTRAN ainsi qu'un langage de manipulation associé. L'architecture globale est représentée figure II.12.

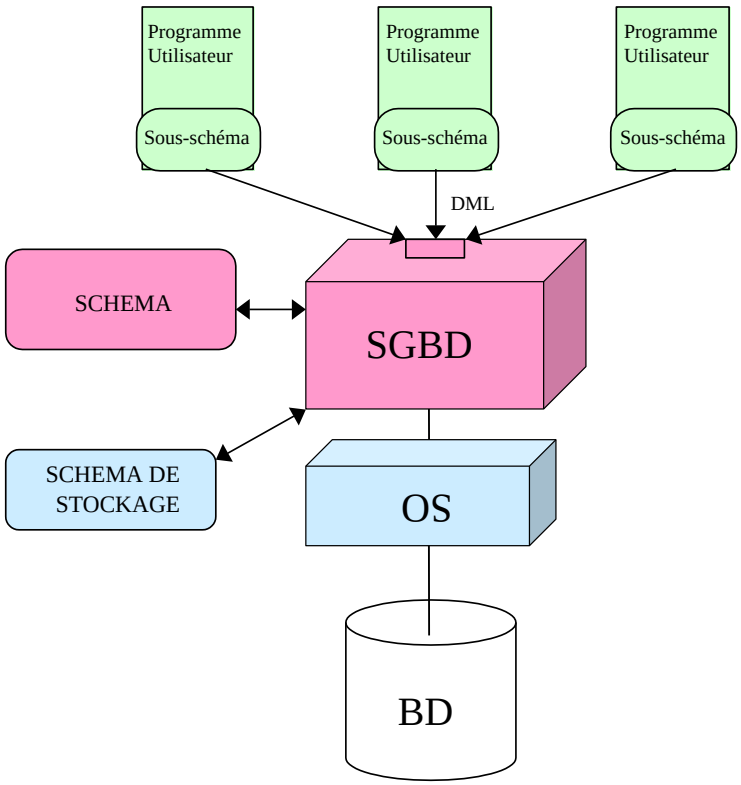

Figure II.12 — Architecture du DBTG CODASYL

6. ARCHITECTURES OPÉRATIONNELLES DES SGBD

Depuis le milieu des années 80, les SGBD fonctionnent selon l'architecture client-serveur. Nous introduisons ces architectures brièvement ci-dessous.

6.1 Les architectures client-serveur

D'un point de vue opérationnel, un SGBD est un ensemble de processus et de tâches qui supportent l'exécution du code du SGBD pour satisfaire les commandes des utilisateurs. Depuis l'avènement des architectures distribuées autour d'un réseau local, les systèmes sont organisés selon l'architecture client-serveur. Cette architecture a été ébauchée dans un rapport du sous-groupe de l'ANSI/X3/SPARC appelé DAFTG (*Database Architecture Framework Task Group*) [ANSI86] et mise à la mode à la fin des années 80 par plusieurs constructeurs de SGBD.

L'**architecture client-serveur** inclut le noyau d'un SGBD tel que décrit ci-dessus, appelé DMCS (Description Manipulation and Control Sub-system), qui fonctionne en mode serveur. Autour de ce serveur s'articulent des processus attachés aux utilisateurs supportant les outils et les interfaces externes. Le DMCS est construit sur le gestionnaire de fichiers ou de disques virtuels du système opératoire. La figure II.13 illustre cette architecture.

Notion II.25 : Architecture client-serveur (*Client-server architecture*)

Architecture hiérarchisée mettant en jeu d'une part un serveur de données gérant les données partagées en exécutant le code du SGBD avec d'éventuelles procédures applicatives, d'autre part des clients pouvant être organisés en différents niveaux supportant les applications et la présentation, et dans laquelle les clients dialoguent avec les serveurs via un réseau en utilisant des requêtes de type question-réponse.

Le langage DL (*Data Language*) est le langage standard d'accès au SGBD, supporté par un protocole de niveau application pour le fonctionnement en mode réparti, appelé **protocole d'accès aux données distantes** (*Remote Data Access RDA*). Ce protocole est aujourd'hui en bonne voie de standardisation. Il est d'ailleurs complété par un protocole de gestion de transactions réparties.

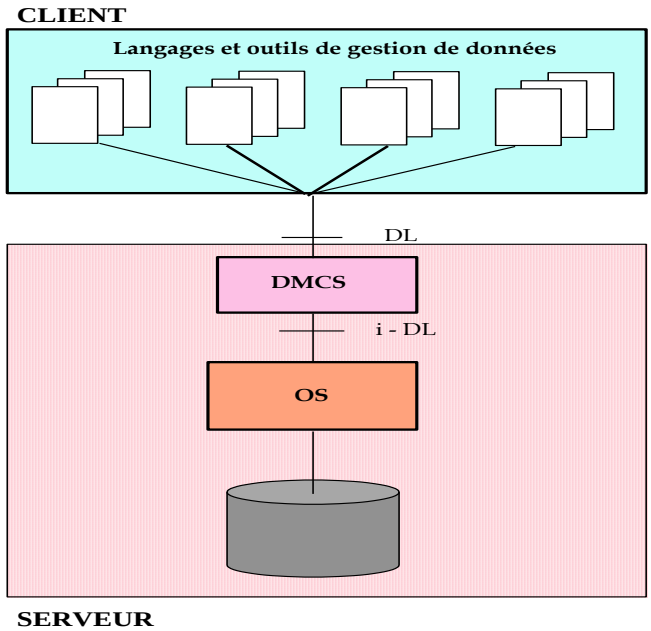

Figure II.13 — L'architecture client-serveur

Il existe différentes variantes de l'architecture client-serveur, selon qu'un processus serveur est associé à chaque utilisateur, ou que plusieurs utilisateurs partagent un même processus serveur. Dans le premier cas, le serveur est monotâche. Chaque processus client a un processus serveur associé. La machine supportant les serveurs doit partager son temps entre eux. Les commutations de processus peuvent être lourdes (quelques millisecondes sur UNIX). De plus, les processus serveurs partageant les mêmes données, il est nécessaire de les synchroniser, par exemple par des sémaphores, afin d'éviter les problèmes de concurrence d'accès. Cela peut entraîner des pertes de temps importantes, et donc de mauvaises performances en présence d'usagers multiples.

Aujourd'hui, plusieurs systèmes proposent un serveur multitâche, capable de traiter plusieurs requêtes d'utilisateurs différents en parallèle. Cela est réalisé grâce à un multiplexage du serveur en tâches, les commutations de tâches étant assurées par le serveur lui-même, au

niveau d'un gestionnaire de tâches optimisé pour les bases de données. Une telle architecture multitâche (en anglais, *multi-thread*) permet de meilleures performances en présence d'un nombre important d'utilisateurs.

Au-delà du *multi-thread*, le besoin en performance a conduit à partionner les traitements applicatifs de façon à réduire les communications entre client et serveur. Ainsi, le client peut invoquer des procédures applicatives qui manipulent la base directement sur le serveur. Ces procédures applicatives liées à la base sont appelées des **procédures stockées**. Elles évitent de multiples commandes et transferts de données sur le réseau. Ceux-ci sont remplacés par l'invocation de procédures stockées avec quelques paramètres et la transmission des paramètres retour. L'architecture obtenue permettant deux couches de traitement applicatifs est appelée **architecture à deux strates** (*two-tiered architecture*).

> **Notion II.26 : Architecture client-serveur à deux strates (*Two-tiered client-server architecture*)**
>
> Architecture client-serveur composée : (i) d'un serveur exécutant le SGBD et éventuellement des procédures applicatives ; (ii) de clients exécutant le corps des applications et la présentation des données.

La figure II.14 propose une vue plus détaillée d'une architecture client-serveur à deux strates. L'application est écrite à l'aide d'un outil applicatif, souvent un L4G. Elle soumet ses demandes de services (requêtes au SGBD ou invocation de procédures stockées au *middleware* qui les transfert au serveur. Le middleware comporte un composant client et un composant serveur qui permettent les échanges de commandes via le protocole réseau. Sur la figure, il est appelé outil de connectabilité. Si vous souhaitez en savoir plus sur le *middleware*, reportez-vous à [Gardarin97].

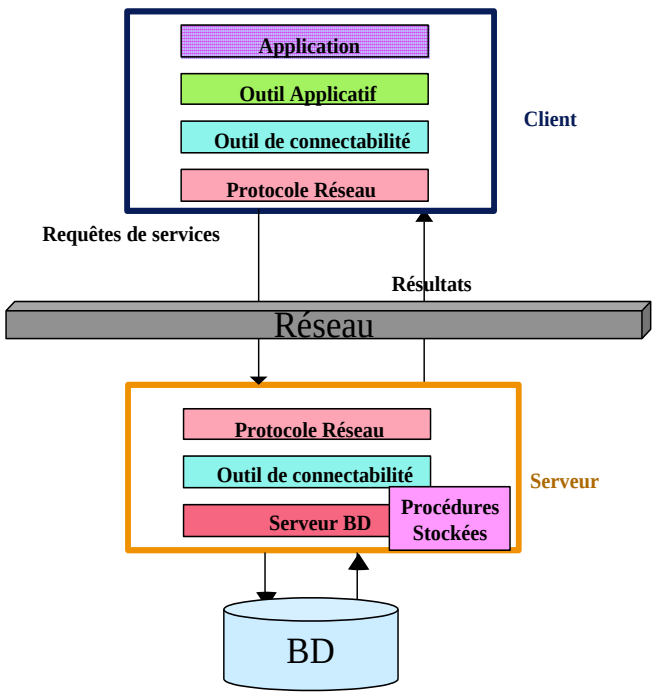

Figure II.14 — L'architecture client-serveur à deux strates

Avec l'apparition d'Internet et du Web, les architectures client-serveur ont évolué vers des **architectures à trois strates** (*three-tiered architecture*). Le client est responsable de la présentation. Il utilise pour cela des *browsers* Web. Le serveur d'application exécute le code applicatif essentiel. Le serveur de données supporte le SGBD et gère éventuellement des procédures stockées.

> **Notion II.27 : Architecture client-serveur à trois strates (*Three-tiered client-server architecture*)**
>
> Architecture client-serveur composée : (i) d'un serveur de données exécutant le SGBD et éventuellement des procédures applicatives ; (ii) d'un serveur d'application exécutant le corps des applications ; (iii) de clients responsables des dialogues et de la présentation des données selon les standards du Web.

La figure II.15 illustre une telle architecture. Les *middlewares* mis en jeu sont beaucoup plus complexes.

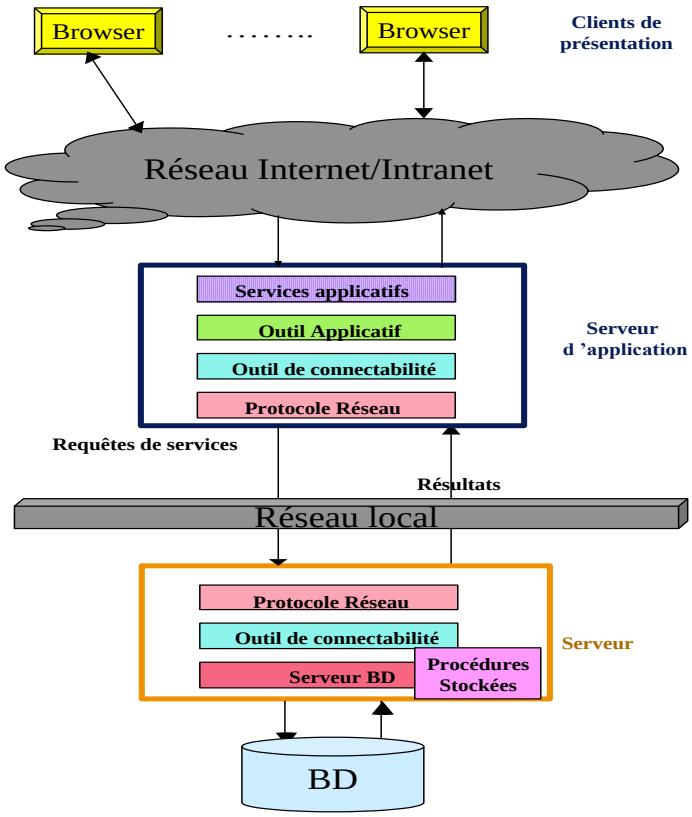

Figure II.15 — L'architecture client-serveur à trois strates

L'architecture client-serveur est aujourd'hui bien adaptée aux systèmes répartis autour d'un réseau local et/ou d'Internet. Elle permet à de multiples postes ou stations de travail distribués sur la planète de partager les mêmes données. Celles-ci sont gérées de manière fiable et avec

de bons contrôles de concurrence au niveau du serveur. Un processus client sur la station de travail ou l'ordinateur personnel gère les applications de l'utilisateur qui émettent des requêtes au serveur. Un client peut même invoquer plusieurs serveurs : on parle alors d'architecture client-multiserveur. L'inconvénient mais aussi l'avantage du client-serveur est de centraliser la gestion des données au niveau du serveur.

6.2 Les architectures réparties

Afin de répondre à la tendance centralisatrice de l'approche client-serveur, certains SGBD préconisent une **architecture répartie**. Une architecture répartie fait interagir plusieurs serveurs gérant un ensemble de bases perçu comme une seule base par les utilisateurs.

> ***Notion : Architecture BD répartie (Distributed database architecture)***
>
> Architecture composée de plusieurs serveurs coopérant à la gestion de bases de données composées de plusieurs sous-bases gérées par un seul serveur, mais apparaissant comme des bases uniques centralisées pour l'utilisateur.

La figure II.16 illustre une architecture répartie. Celle-ci est composée de différents serveurs munis de SGBD différents et spécialisés. C'est un exemple de base de données répartie hétérogène, encore appelée base de données fédérée. Nous n'étudions pas les bases de données réparties dans cet ouvrage. L'auteur pourra se reporter à [Gardarin97] et à [Valduriez99] pour une étude plus complète des systèmes de bases de données réparties.

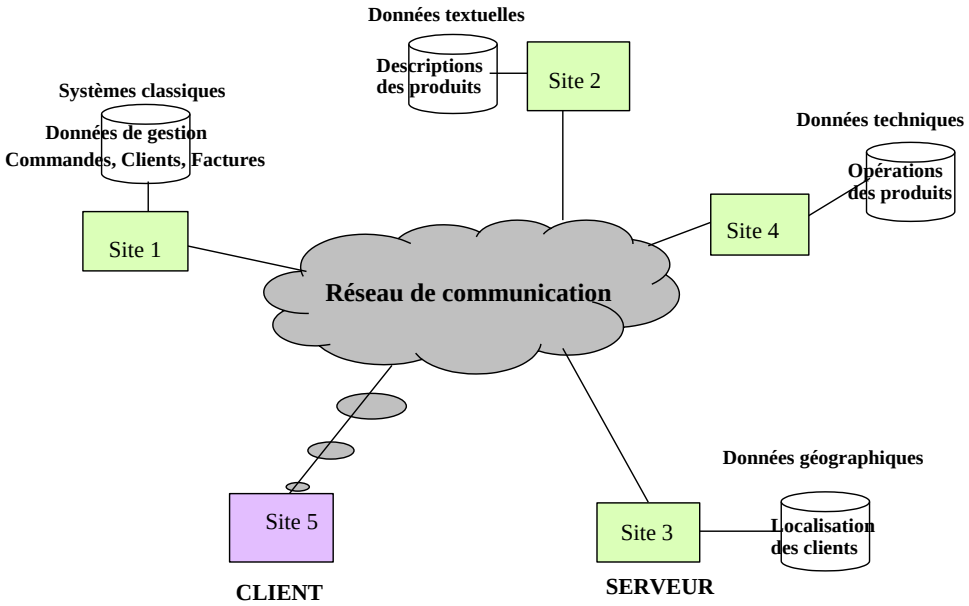

Figure II.16 — Exemple d'architecture répartie

7. CONCLUSION

Dans ce chapitre, nous avons présenté les objectifs des systèmes de gestion de bases de données, puis les concepts et méthodes essentiels de ces systèmes. Cela nous a conduit à étudier les architectures fonctionnelles, puis les architectures opérationnelles. Afin de concrétiser nos propos, nous avons introduit une variante du langage QUEL pour décrire et manipuler les données. L'architecture proposée en 1978 par le groupe ANSI/X3/SPARC permet de bien comprendre les niveaux de schémas possible. L'architecture fonctionnelle de référence est voisine de celle retenue dans le système SABRINA [Gardarin86] ou encore de celle des systèmes INGRES ou ORACLE. Les architectures opérationnelles sont aujourd'hui client-serveur, mais évoluent de plus en plus souvent vers le réparti.

Il est possible de résumer ce chapitre en rappelant qu'un SGBD offre une interface de description de données qui permet de documenter le dictionnaire de données. Le compilateur du langage de description gère cette métabase. Un SGBD offre aussi une interface de manipulation de données (recherches et mises à jour) qui permet de modifier ou de retrouver des données dans la base. Le compilateur-optimiseur du langage de manipulation génère des plans d'accès optimisés. Ceux-ci sont exécutés par le processus serveur, qui gère aussi la concurrence et la fiabilité. Les requêtes peuvent être émises par des programmes d'applications écrits dans des langages plus ou moins traditionnels, ou par des utilisateurs travaillant en interactif à partir d'utilitaires.

Les SGBD tels que présentés dans ce chapitre s'appuient au niveau interne sur une gestion de fichiers. Celle-ci peut être une partie intégrante du système opératoire. Elle peut être plus ou moins sophistiquée. Dans le chapitre suivant, nous allons étudier la gestion de fichiers, de la plus simple à la plus élaborée. Selon le niveau de sophistication du gestionnaire de fichiers sur lequel il est construit, un SGBD devra ou non intégrer des fonctionnalités de type gestion de fichiers au niveau interne. Aujourd'hui, la plupart des SGBD intègrent leurs propres méthodes d'accès aux fichiers (parfois appelés segments), du type de celles que nous allons étudier dans le chapitre suivant.

8. BIBLIOGRAPHIE

[ANSI75] ANSI/X3/SPARC Study Group on Data Base Management Systems, "Interim Report", *ACM SIGMOD Bulletin*, Vol. 7, N° 2, ACM Ed., 1975.

Ce document présente l'architecture ANSI/X3/SPARC et ses trois niveaux de schémas.

[ANSI78] ANSI/X3/SPARC Study Group on Data Base Management Systems, "Framework Report on Database Management Systems", *Information Systems*, Vol. 3, N° 3 1978.

Il s'agit du document final présentant l'architecture à trois niveaux de schémas de l'ANSI.

[ANSI86] ANSI/X3/SPARC Database Architecture Framework Task Group, "Reference Model for DBMS Standardization", *ACM SIGMOD Record*, Vol. 15, N° 1, mars 1986.

Il s'agit du document final présentant l'étude sur les architectures de SGBD du groupe DAFTG. Celui-ci conseille de ne pas chercher à standardiser davantage les architectures, mais plutôt de s'efforcer de standardiser les langages.

[Astrahan76] Astrahan M., Blasgen M., Chamberlin D., Eswaran K., Gray. J., Griffiths P., King W., Lorie R., McJones P., Mehl J., Putzolu G., Traiger I., Wade B., Watson V., "System R: Relational Approach to Database Management", *ACM Transactions on Database Systems*, Vol. 1, N° 2, juin 1976.

*Cet article présente System R, le premier prototype de SGBD relationnel réalisé par IBM dans son centre de recherches de San José. System R comporte une architecture à deux niveaux de schéma et deux processeurs essentiels : le RDS (*Relational Data System*) qui comprend analyseur, traducteur et optimiseur, et le RSS qui correspond à l'exécuteur. System R a donné naissance à SQL/DS et à DB2.*

[Brodie86] Brodie M. Ed., *On Knowledge Base Management Systems*, Springer Verlag Ed., Berlin, 1986.

Ce livre discute les problèmes liés à l'introduction des techniques de l'intelligence artificielle au sein des SGBD. Les conséquences en termes d'objectifs et d'architecture sont analysées tout au long des multiples articles composant le livre.

[Carey85] Carey M.J., Dewitt D.J., "Extensible Database Systems", *Islamodrada Workshop*, 1985, dans [Brodie86].

Cet article plaide pour une architecture extensible des SGBD, notamment au niveau de l'optimiseur. Il propose de réaliser un optimiseur dirigé par une bibliothèque de règles spécifiant les techniques d'optimisation.

[Codasyl71] CODASYL DBTG, *Codasyl Data Base Task Group Report*, ACM Ed., New-York, avril 1971.

Ce document présente la synthèse des travaux du CODASYL en termes d'architecture, de langage de description et de langage de manipulation de données pour SGBD supportant le modèle réseau. Le groupe CODASYL était une émanation du groupe de standardisation de COBOL. Il a tenté de standardiser la première génération de SGBD.

[Chen76] Chen P.P., "The Entity-Relationship Model - Towards a Unified View of Data", *ACM Transactions on Database Systems*, Vol. 1, N° 1, Mars 1976.

Cet article introduit le modèle entité-association pour décrire la vue des données d'une entreprise. En particulier, les diagrammes de Chen sont présentés. Il est montré que le modèle permet d'unifier les différents points de vue.

[Date71] Date C.J., Hopewell P., "File Definition and Logical Data Independance", *ACM SIGFIDET Workshop on Data Description, Access and Control*, ACM Ed., New-York, 1971.

L'indépendance physique et logique est discutée clairement pour la première fois. En particulier, les modifications de structures de fichiers qui doivent être possibles sans changer les programmes sont analysées.

[Gardarin86] Gardarin G.,Kerhervé B., Jean-Noël M., Pasquer F., Pastre D., Simon E., Valduriez P., Verlaine L., Viémont Y., "SABRINA: un système de gestion de bases de données relationnel issu de la recherche", *Techniques et Sciences Informatique (TSI)*, Dunod Ed., Vol. 5, N° 6, 1986.

Cet article présente le SGBD relationnel SABRINA réalisé dans le projet SABRE à l'INRIA, puis en collaboration avec plusieurs industriels, de 1979 à 1989. Ce système avancé supporte une architecture extensible voisine de l'architecture de référence présentée ci-dessus. Il est construit selon l'approche client-serveur.

[Gardarin97] Gardarin G., Gardarin O., *Le Client-Serveur*, 470 pages, Editions Eyrolles, 1997.

Ce livre traite des architectures client-serveur, des middlewares et des bases de données réparties. Les notions importantes du client-serveur sont clairement expliquées. Une part importante de l'ouvrage est consacrée aux middlewares et outils de développement objet. CORBA et DCOM sont analysés. Ce livre est un complément souhaitable au présent ouvrage, notamment sur les bases de données réparties et les techniques du client-serveur.

[Stonebraker74] Stonebraker M.R., "A Functional View of Data Independance", *ACM SIGMOD Workshop on Data Description, Access and Control*, ACM Ed., mai 1974.

Un des premiers articles de Mike Stonebraker, l'un des pères du système INGRES. Il plaide pour l'introduction de vues assurant l'indépendance logique.

[Stonebraker76] Stonebraker M., Wong E., Kreps P., Held G.D., "The Design and Implementation of Ingres", *ACM Transactions on Database Systems*, Vol. 1, N° 3, septembre 1976.

Cet article décrit le prototype INGRES réalisé à l'université de Berkeley. Celui-ci supporte le langage QUEL. Il est composé de 4 processus correspondant grossièrement à l'analyseur, au traducteur, à l'optimiseur et à l'exécuteur présenté ci-dessus. Ce prototype est l'ancêtre du système INGRES aujourd'hui très populaire.

[Stonebraker80] Stonebraker M., "Retrospection on a Database system", *ACM Transactions on Database Systems*, Vol.5, N° 2, mars 1980.

Cet article fait le bilan de la réalisation du système INGRES. Il souligne notamment l'importance du support génie logiciel pour la réalisation d'un SGBD. Par exemple, lorsqu'un module change de responsable, le nouveau responsable s'empresse de le réécrire sous pretexte de non propreté, etc.

[Stonebraker87] Stonebraker M., "The Design of the POSTGRES Storage System", *Int. Conf. on Very Large Databases*, Morgan & Kauffman Ed., Brighton, Angleterre, 1987.

M. Stonebraker présente la conception du noyau de stockage du système POSTGRES, successeur d'INGRES. Celui-ci était entièrement basé sur le contrôle de concurrence et le support de déclencheurs.

[Tsichritzis78] Tsichritzis D.C., Klug A. (Editeurs), *The ANSI/X3/SPARC Framework*, AFIPS Press, Montvale, NJ, 1978.

Une autre version du rapport final de l'ANSI/X3/SPARC avec ses trois niveaux de schémas.

[Valduriez99] Valduriez P., Ozsu T., *Principles of Distributed Database Systems*, 562 pages, Prentice Hall, 2e édition,1999.

Le livre fondamental sur les bases de données réparties en anglais. Après un rappel sur les SGBD et les réseaux, les auteurs présentent l'architecture type d'un SGBD réparti. Ils abordent ensuite en détails les différents problèmes de conception d'un SGBD réparti : distribution des données, contrôle sémantique des données, évaluation de questions réparties, gestion de transactions réparties, liens avec les systèmes opératoires et multibases. La nouvelle édition aborde aussi le parallèlisme et les middlewares. Les nouvelles perspectives sont enfin évoquées.

[Weldon79] Weldon J.L., "The Practice of Data Base Administration", *National Computer Conference*, AFIPS Ed., V.48, New York, 1979.

Cet article résume les résultats d'une étude du rôle des administrateurs de données à travers une enquête réalisée auprès de 25 d'entre eux. Un appendice permet plus particulièrement de définir leurs tâches : établissement du schéma directeur, conception des bases de données, tests, contrôles et support opérationnel.

[Zaniolo83] Zaniolo C., "The Database Language GEM", *ACM SIGMOD Int. Conf. on Management of Data*, San José, CA, 1983.

Cet article présente une extension très complète du langage QUEL pour le modèle entité-association. Cette extension a été implantée au-dessus d'une machine bases de données exécutant un langage proche de QUEL.

[Zook77] Zook W. *et. al.*, *INGRES Reference Manual*, Dept. of EECS, University of California, Berkeley, CA, 1977.

Ce document décrit les interfaces externes de la première version d'INGRES et plus particulièrement le langage QUEL.