Université de Nice Sophia-Antipolis

Introduction aux bases de données, modèle relationnel

Version 2.0

Richard Grin

26 janvier 1999

Table des matières

Présentation du polycopié

Ce polycopié présente quelques notions sur les bases de données, en introduction à un cours sur le langage SQL.

La première partie donne quelques généralités sur les bases de données et sur les méthodes de développement des applications. La complexité de la plupart des applications rend impossible l'écriture directe du code qui sera exécuté par les ordinateurs. Cette partie présente rapidement les tâches qui précèdent l'écriture de ce code.

Plusieurs méthodes de développement sont utilisées par les développeurs. Elles s'appuient sur des modèles pour représenter les traitements et les données. Cette première partie présentera le modèle entité-association, utilisé par de nombreuses méthodes non orientées objet, et la notation UML, introduite pour décrire les données et les traitements des applications orientées objets.

La deuxième partie présente le modèle relationnel qui est, à ce jour, le modèle dominant pour représenter les données manipulées par les SGDB. Elle montre comment traduire un schéma entité-association et un diagramme de classes (au sens du modèle objet) en schéma relationnel.

La troisième partie présente quelques notions utiles pour le développeur, liées à l'implantation et à l'utilisation des SGBD relationnels.

Des illustrations pratiques seront données dans les travaux dirigés et dans la deuxième partie du cours, consacrée au langage SQL.

Ce cours n'est qu'un rapide survol de ces notions de base. Le lecteur pourra approfondir ces notions, s'il le juge nécessaire, en consultant des livres ou des sites Web.

Pour plus de précisions sur le modèle relationnel, le lecteur pourra consulter, par exemple, le livre de Serge Miranda "Comprendre et concevoir les bases de données relationnelles" aux éditions PSI (ou à défaut, le livre de Serge Miranda et José-Maria Busta "L'art des bases de données" tomes 2 et 3 aux éditions Eyrolles), le livre de Georges Gardarin "Bases de données ; les systèmes et leurs langages" aux mêmes éditions ou le livre "Des structures aux bases de données" de Christian Carrez aux éditions Dunod Informatique (pour les notions sur les formes normales).

Pour UML, on peut conseiller la lecture du livre "Modélisation objet avec UML" de Pierre-Alain Muller aux éditions Eyrolles et la consultation du site Web `http://www.essaim.univ-mulhouse.fr/uml/` (voyez en particulier le lien "Contributions").

Le site de référence pour UML est en anglais: `http://www.rational.com/uml/`. Vous y trouverez en particulier un lien vers le livre écrit par les auteurs d'UML: "The Unified Modeling Language User Guide" (Addison-Wesley Object Technology Series).

Les remarques et les corrections d'erreurs peuvent être envoyées par courrier électronique à l'adresse `grin@unice.fr`, en précisant le sujet "Poly Bases de données".

Première partie

Généralités sur les bases de données

Chapitre 1

Définition et fonctions d'un SGBD

Un essai de définition: une base de données est un ensemble structuré de données enregistrées dans un ordinateur et accessibles de façon sélective par plusieurs utilisateurs.

Un logiciel qui permet d'interagir avec une base de données s'appelle un système de gestion de base de données (S.G.B.D.).

1.1 Fonctions d'un SGBD

Un SGBD doit permettre de:

- décrire les données qui seront stockées,
- manipuler ces données (ajouter, modifier, supprimer des informations),
- obtenir des renseignements à partir de ces données (sélectionner, trier, calculer, agréger, etc.),
- définir des contraintes d'intégrités sur les données (contraintes de domaines, d'existence, etc.),
- définir des protections d'accès (mots de passe, autorisations, etc.),
- résoudre les problèmes d'accès multiples aux données (blocages, interblocages),
- prévoir des procédures de reprise en cas d'incident (sauvegardes, journaux, etc.).

De plus, un SGBD doit permettre d'écrire des applications indépendantes de l'implantation physique des données (codage, ordre dans lequel les données sont enregistrées, support d'enregistrement, etc.).

1.2 Avantages de l'utilisation des SGBD

Avant l'arrivée des bases de données (et encore dans beaucoup de cas aujourd'hui), chaque application écrite pour un organisme travaillait avec ses propres fichiers.

Une même information (l'adresse d'un client par exemple) peut alors être enregistrée dans plusieurs fichiers disjoints. Ceci occasionne des délais de mise à jour et peut amener les différentes applications à travailler sur des données contradictoires.

Quand la gestion se fait avec une base de données centralisée (centralisée "logiquement" mais pas nécessairement physiquement si la base de données est répartie sur plusieurs sites), chaque donnée n'est enregistrée qu'en un seul endroit de la base et il ne peut y avoir ce genre de problèmes. Cette centralisation facilite donc le maintien de l'intégrité des données.

Les facilités offertes par les SGBD pour contrôler l'accès des utilisateurs aux données de la base et les reprises automatisées après incident accroissent la sécurité dans le traitement des données.

Les SGBD offrent aussi des instructions très puissantes pour le traitement des données: un seul ordre "select" du langage SQL peut correspondre à des dizaines de lignes de programmation dans un langage de troisième génération comme le langage C. La productivité des programmeurs est ainsi fortement augmentée.

En plus de l'indépendance des programmes vis-à-vis de l'implantation physique des données, d'autres avantages importants sont apportés par l'utilisation des SGBD évolués des dernières générations (en particulier par les SGBD relationnels). Ces SGBD offent:

- l'indépendance des programmes vis-à-vis de la structure logique des données (stratégie d'accès aux données, manière de regrouper les données, etc.),
- l'interrogation directe de la base par les utilisateurs dans un langage non procédural.

Ces différents points facilitent grandement la maintenance des applications et permettent plus de souplesse pour le traitement des données enregistrées.

1.3 Niveaux de description des bases de données

La norme ANSI/SPARC définit 3 niveaux pour décrire l'architecture d'une base de données:

- le *niveau externe* correspond aux différentes façons dont les utilisateurs peuvent voir les données de la base. La représentation de la base est donc composée de plusieurs schémas externes.
- le *niveau conceptuel* correspond à une vision globale de la base. Il n'y a qu'un seul schéma conceptuel. Les données sont décrites ainsi que les contraintes d'intégrité liées aux données.
- le *niveau interne* correspond à la manière dont la base est implantée sur les ordinateurs (description des enregistrements contenant les données, des index, etc.)

Ce découpage facilite la maintenance des applications:

- il permet de dissocier autant que possible les applications des contraintes liées au matériel ou au système d'exploitation (seul le niveau interne en dépend),
- la séparation entre les niveaux externe et conceptuel permet de protéger les applications envers les modifications ou enrichissement du schéma conceptuel.

Ce cours concerne plus particulièrement le niveau conceptuel. Le chapitre 7 aborde des notions liées au niveau interne.

1.4 Types d'utilisateurs d'une base de données

L'*administrateur* de la base est chargé du contrôle de la base de données. Il est chargé de permettre l'accès aux données aux applications ou individus qui y ont droit et de conserver de bonnes performances d'accès à ces données. Il est aussi chargé des sauvegardes et des procédures de reprise après les pannes.

Le *programmeur* d'applications utilise la base de données pour construire ses applications. Il a droit de créer de nouvelles tables et les structures associées (vues, index, cluster, etc.). Il définit avec l'administrateur de la base les droits qui seront accordés aux utilisateurs des applications qu'il développe.

L'*utilisateur final* n'a accès qu'aux données qui lui sont utiles. L'administrateur de la base peut lui accorder certains droits : consultations, modification, suppression des données. En général il n'a pas le droit de créer de nouvelles tables ni d'ajouter ou d'enlever des index.

Chapitre 2

Types de SGBD

Historiquement les premiers SGBD étaient de type hiérarchique, puis sont apparus les SGBD de type réseau. Actuellement, la plupart des nouveaux systèmes sont de type relationnel. Des SGBD de type "objet" commencent à apparaitre sur le marché et ils remplaceront vraisemblablement une partie des SGBD relationnels.

La différence essentielle entre ces types de SGBD réside dans les modèles sur lesquels ils s'appuient pour représenter les données.

2.1 Modèle hiérarchique

Les données sont représentées sous forme d'une structure arborescente d'enregistrements. Cette structure est conçue avec des pointeurs et détermine le chemin d'accès aux données.

2.2 Modèle réseau

La structure des données peut être visualisée sous la forme d'un graphe quelconque. Comme pour le modèle hiérarchique, la structure est conçue avec des pointeurs et détermine le chemin d'accès aux données.

Pour les modèles hiérarchique et réseau, les programmes ne sont pas indépendants de la structure logique de la base et du chemin d'accès aux données : ils doivent décrire comment retrouver les données (on parle de *navigation* dans la base) et si, par exemple, on enlève un index, tous les programmes qui l'utilisaient doivent être réécrits. De plus le langage de travail est complexe.

2.3 Modèle relationnel

Il est fondé sur la théorie mathématique des relations.

Il conduit à une représentation très simple des données sous forme de tables constituées de lignes et de colonnes. Il n'y a plus de pointeurs qui figeaient la structure de la base.

La souplesse apportée par cette représentation et les études théoriques appuyées sur la théorie mathématique des relations ont permis le développement de langages puissants non procéduraux. Dans ces langages, l'utilisateur ou le programmeur indique quelles informations il veut obtenir et c'est le SGBD qui trouve la manière d'arriver au résultat. Le programme ou l'utilisateur n'a plus à naviguer dans la base pour retrouver ses données. Ces langages peuvent être utilisés par des non-informaticiens et permettent l'écriture de programmes indépendants de la structure logique et physique des données.

Le langage SQL est un standard parmi tous ces langages.

L'inventeur du modèle relationnel est Codd qui travaillait dans les laboratoires d'IBM. Il a énoncé douze règles que doivent vérifier les SGBD pour être relationnels. En fait, aucun SGBD ne respecte actuellement toutes ces règles. Certains cependant s'approchent de cette "perfection" relationnelle. Voici ces douze règles (certaines notions comme les vues et les transactions seront développées plus loin dans ce cours ou dans le cours sur le langage SQL) :

Règle 1 : toutes les informations sur les données sont représentées au niveau logique et non physique (pas besoin d'informations sur la façon dont sont enregistrées physiquement les données)

Règle 2 : les données sont accessibles uniquement par la combinaison du nom de la table, de la clé primaire (voir 4.1.2) et du nom de la colonne (pas de chemin à donner)

Règle 3 : une valeur spéciale doit représenter l'absence de valeur (valeur NULL)

Règle 4 : la description de la base de données doit être accessible comme les données ordinaires (un dictionnaire des données est enregistré dans la base)

Règle 5 : un langage doit permettre de définir les données, définir des vues (visions particulières de la base, enregistrées comme des relations), manipuler les données, définir des contraintes d'intégrité, des autorisations, de gérer des transactions (voir 6.1.1)

Règle 6 : on peut faire des mises à jour par les vues lorsque c'est logiquement possible

Règle 7 : le langage doit comporter des ordres effectuant l'insertion, la mise à jour et la suppression de données (un seul ordre pour effectuer chacune de ces fonctions)

Règle 8 : indépendance des programmes vis-à-vis de l'implantation physique des données

Règle 9 : indépendance des programmes vis-à-vis de l'implantation logique des données (si les informations manipulées par les programmes n'ont pas été modifiées ou supprimées)

Règle 10 : les contraintes d'intégrité doivent pouvoir être définies dans le langage relationnel et enregistrées dans le dictionnaire des données

Règle 11 : indépendance vis à vis de la répartition des données sur divers sites

Règle 12 : on ne peut jamais contourner les contraintes (d'intégrité ou de sécurité) imposées par le langage du SGBD en utilisant un langage de plus bas niveau (par exemple le langage C).

2.4 Modèle objet

Les données sont représentées sous forme d'objets au sens donné par les langages orientés objet : pour simplifier, les données (au sens habituel) sont enregistrées avec les procédures et fonctions qui permettent de les manipuler. Les SGBD orientés objet (SGBDOO) supportent aussi la notion d'héritage entre classes d'objets.

Le développement des langages orientés objet a suscité le besoin de conserver les objets manipulés par ces langages. De plus les manipulations de données à structures complexes, en particulier dans les traitements qui font intervenir le multimedia, sont facilitées avec les SGBDOO. Le modèle relationnel a montré ses limites pour ce type de données.

Cependant ces SGBD ne sont utilisés actuellement que pour des usages bien spécifiques et il n'existe pas encore de normes communément admises. Pour cette raison, la suite de ce cours portera sur le modèle relationnel qui, de toute façon, sera encore utilisé de nombreuses années.

La recherche est active dans le domaine des SGBDOO. On peut s'attendre à leur développement rapide, ne serait-ce que pour enregistrer les objets manipulés par les langages objets comme Java dont le taux de pénétration dans les entreprises est rapide. Pour garder leur part du marché, les grands SGBD relationnels ajoutent progressivement une couche objet à leur noyau relationnel. Qui dominera demain le marché des SGBD? des nouveaux produits fondés sur le modèle objet ou les anciens SGBD relationnels, renommés "universels", censés convenir aussi bien au relationnel qu'à l'objet ?

Chapitre 3

Développement d'une application

Ce chapitre décrit sommairement les grandes étapes du développement d'une application. Une étude approfondie nécessiterait un cours complet.

L'approche "objet" sera privilégiée. Les toutes dernières années ont en effet vu l'explosion de l'utilisation des langages objets pour le développement des applications, surtout grâce à l'émergence du langage Java. Les anciennes méthodes de développement, comme Merise (méthode très utilisée dans le monde francophone dans les années 80 et jusqu'à aujourd'hui), sont progressivement abandonnées, malgré des tentatives d'adaptation au monde de l'objet. Ces méthodes s'appuient le plus souvent sur le modèle de données "entité-association". Elles sont remplacées par de nouvelles méthodes mieux adaptées aux objets.

À ce jour, aucune méthodologie de développement objet ne s'est vraiment dégagée comme un standard. Cependant la notation UML est adoptée par le plus grand nombre. UML (*Unified Modeling Language*) propose une manière de décrire, le plus souvent graphiquement, les résultats des différentes étapes du développement d'une application. Mais UML n'indique pas la démarche pour obtenir ces résultats et ne peut donc être considérée comme une méthodologie de développement. Ce chapitre fait un très rapide survol d'UML en décrivant avec un peu plus de précision les diagrammes de classe qui jouent un rôle important dans le passage au modèle relationnel.

Dans le chapitre suivant on montrera comment traduire un schéma entité-association et un diagramme de classe UML en un schéma relationnnel, pour préparer l'implantation dans un SGBD relationnel. Pourquoi ne pas utiliser un SGBDOO ? La logique interne de ce cours (qui mène à l'étude du langage SQL) et le marché actuel des SGBD conduisent à priviligier les SGBD relationnels et non pas les SGBD objets. Quand les SGBD objets seront plus répandus, certaines des étapes décrites à la fin de ce chapitre et dans le chapitre suivant ne seront plus nécessaires, ou seront grandement facilitées.

3.1 Étapes dans le développement d'une application

Lorsqu'une application doit être développée, de nombreuses étapes vont précéder l'écriture des lignes de code et la saisie des données utilisées par l'application.

Le processus global est un processus descendant. On part d'un haut niveau d'abstraction et on entre progressivement dans les détails pour atteindre finalement le niveau où l'on peut implanter l'étude sous la forme d'instructions d'un langage informatique.

3.1.1 Itérations

Cette démarche n'est pas linéaire. Un important problème des anciennes méthodes étaient qu'elles suivaient une démarche trop souvent linéaire, rendant difficiles des retours en arrière ou des raffinements du travail déjà effectué. Les méthodes orientées objet ont formalisé la notion d'itération. Elles ont admis qu'il est très difficile, sinon impossible dès que l'application a un minimum de complexité, de tout prévoir lorsque l'on aborde pour la première fois une étape du développement.

Les raffinements successifs du travail déjà effectué font partie intégrante de la démarche de développement et sont des étapes normales du développement et non pas des rattrapages de négligences. On se fixe des objectifs précis à chaque itération ; il ne s'agit pas de faire de l'à-peu-près en comptant sur les prochaines itérations pour améliorer la solution. On commence par étudier les parties les plus importantes de l'application, et celles qui présentent le plus de risques pour la faisabilité d'un choix de conception ou d'architecture. La démarche est incrémentale : à chaque itération on ajoute de nouvelles fonctionnalités ou/et on affine le fonctionnement de l'application.

Il est important de noter que chaque itération entraîne la livraison d'un prototype exécutable qui va permettre aux utilisateurs de tester ce qui a été fait, et aux développeurs de vérifier la faisabilité des points délicats à risques.

3.1.2 Phases du développement

Le développement va passer essentiellement par 4 grandes phases qui constituent ce que l'on appelle le *cycle de vie* de l'application :

Spécification des besoins : elle permet de comprendre ce qui est demandé, quels sont les besoins des futurs utilisateurs de l'application.

Cette étape est aussi l'occasion de se familiariser (si ce n'est déjà fait) avec le domaine à informatiser. On détermine les principaux intervenants et les principales activités concernées par l'application.

Analyse : elle construit un modèle de ce qu'il faudra obtenir. Elle ne décrit pas *comment* on obtiendra ce résultat.

Elle permet de comprendre les structures impliquées dans l'application, les échanges de données entre ces structures, les principaux processus liés aux actions à informatiser et les objectifs précis à atteindre.

Conception : elle invente une solution pour informatiser le domaine étudié en respectant les objectifs fixés. Cette étape introduit le "comment", mais à un niveau conceptuel, en essayant d'éviter de faire intervenir les problèmes liés au matériel et au système d'exploitation. On précise l'architecture logicielle de l'application (important, par

exemple, pour les applications distribuées) ; on décrit les traitements et la structure des données.

Implémentation : son résultat est l'écriture des programmes qui vont exécuter les processus définis dans l'étape précédente, l'enregistrement des données dans la base de données et le lancement de la nouvelle application.

Chaque itération va passer par ces phases. Les premières itérations vont certainement passer plus de temps sur les premières phases (surtout la spécification des besoins) que les itérations suivantes.

La taille d'une itération ne peut être facilement déterminée. Elle dépend du type de l'application. Sur une grosse application (18 mois à 2 ans de développement), on peut tabler sur une itération tous les 2 ou 3 mois. On peut cependant très bien avoir un cycle plus rapproché sur de plus petites applications. Dans ce cas, on peut avoir au moins une première itération pour tester les choix d'architecture et évaluer les risques pour la faisabilité, et une seconde pour terminer l'application.

Deux principes à respecter :

- accompagner toutes ces phases d'une documentation. Cette documentation sera utilisée lors des étapes ultérieures du développement, lors de l'utilisation et de la maintenance de l'application. Il est en particulier conseillé de ne pas écrire le guide de l'utilisateur au dernier moment mais plutôt au fur et à mesure du développement de l'application. Les outils de développement fournissent souvent une documentation automatique qui constitue le noyau de la documentation finale.
- consulter et obtenir l'accord des utilisateurs pour toutes les décisions importantes prises lors du développement. Un moment essentiel pour obtenir l'avis et l'accord des utilisateurs est la présentation des prototypes conçus par une itération. Attention à l'explosion (et aux modifications) des besoins exprimés par les utilisateurs quand on leur montre ces prototypes ! Ce problème est très fréquent et souvent difficile à gérer pour le développeur.

3.2 Niveaux de conception

Le développement d'une application va conduire à décrire les données et les traitements suivant plusieurs niveaux. Pour les données[1], les différents niveaux sont :

Niveau conceptuel : Il décrit les données sous une forme indépendante du matériel et du SGBD qui sera utilisé. De nombreux modèles peuvent être utilisés. Ils sont souvent du type "entité-association" ou "objet".

Niveau logique : Il adapte l'étude faite au niveau conceptuel au type de SGBD utilisé. En particulier, pour les SGBD relationnels, il traduit le niveau conceptuel sous forme de relations/tables.

1. qui nous intéressent plus spécialement dans ce cours, mais les études des données et des traitements doivent être étroitement associées lors du développement d'une application

Pour éviter des problèmes de mises à jour liés à une mauvaise répartition des données entre les tables, les relations sont normalisées (voir chapitre 5). Des besoins d'optimisation peuvent conduire ensuite à dénormaliser certaines relations ; le concepteur peut par exemple introduire des redondances de données pour pouvoir satisfaire plus rapidement certains requêtes fréquentes qui engendrent des jointures de tables coûteuses (voir chapitre 4).

Niveau physique : il décrit l'implantation du niveau logique à l'aide du matériel et du SGBD choisi. Les particularités du SGBD sont utilisées pour optimiser la gestion de la base. Sous Oracle (SGBD relationnel), on définit par exemple les index et on peut prévoir la création de *clusters* (voir 6.5). On s'intéresse aussi aux supports sur lesquels seront enregistrées les données dans l'ordinateur (bonne répartition des tables et des index sur les différents disques, par exemple).

Ce découpage en niveaux d'abstraction facilite la tâche des concepteurs qui n'est pas noyé par les détails et les contraintes techniques au début de son étude. Et surtout il facilitera l'adaptation de l'application aux futures modifications de l'environnement. En effet, l'étude du niveau conceptuel sera rarement remise en cause par les modifications les plus fréquemment rencontrées dans les entreprises (changement des ordinateurs ou du SGBD).

3.3 Méthodes et outils

Le développement d'une application est souvent une tâche longue et complexe. De nombreuses méthodes ont été conçues pour aider les développeurs. Elles ne couvrent pas toujours toutes les phases de développement ; elles sont plutôt orientées vers l'analyse et la conception. Elles manquent parfois de rigueur et on se perd un peu dans les nombreuses variantes développées par les sociétés de services qui les utilisent mais elles sont un guide utile pour le concepteur. On peut espérer à l'avenir une harmonisation des différentes méthodes.

La méthode Merise a été longtemps la plus utilisée dans les pays francophones. Elle s'appuie sur le modèle "entité-association" qui est présenté en 3.4. Elle est maintenant lentement abandonnée au profit des méthodes orientées objets, au fur et à mesure de l'utilisation croissante des langages objets tels C++ et surtout Java.

Aucune méthode orientée objet n'est aujourd'hui devenue un standard mais presque toutes les méthodes ont adopté UML pour décrire les résultats de l'étude.

Une utilisation intensive des méthodes de développement nécessitent des outils de développement. En effet, la complexité des données et des relations entre les différentes parties et concepts dépassent les possibilités de l'esprit humain, surtout quand l'application est développée par des équipes qui travaillent en parallèle sur l'application. De nombreux outils sont disponibles dans le commerce. Des versions d'évaluations permettent de les tester, soit pour un temps limité, soit pour un nombre limité d'entités ou d'objets.

Les sections suivantes font un rapide survol des modèles "entité-association" et "objet" utilisés par les méthodes. Nous décrirons tout d'abord le modèle entité-association car il

est encore largement utilisé. Le modèle objet est certainement promis à un bel avenir. Nous introduirons la représentation graphique UML, en insistant essentiellement sur les diagrammes de classes.

3.4 Modèle entité-association

3.4.1 Présentation du modèle entité-association

Le modèle "entité-association" (on dit aussi "entité-relation") est un modèle pour représenter les données d'un système d'information au niveau conceptuel (voir 1.3). Ce modèle repère et décrit des entités, et des associations entre les entités du système étudié.

Sans entrer dans les détails de la théorie de ce modèle, disons qu'une *entité* est un objet concret (par exemple un produit fabriqué par l'entreprise) ou abstrait (par exemple une livraison d'un produit) qui a une existence propre dans le système étudié. Une *association* correspond à l'existence d'une relation entre entités (par exemple, le fait qu'une livraison va concerner tel produit de l'entreprise).

Le *degré* d'une association est le nombre d'entités que l'association relie.

En fait, le modèle "entité-association" va s'intéresser aux *types* d'entités et d'associations plutôt qu'aux entités et associations prises individuellement. Par exemple, tous les produits fabriqués par l'entreprise vont être regroupés sous l'entité générique "PRODUIT" et les associations qui concernent un produit et une livraison vont appartenir au type d'association "LIVRÉ PAR". On parlera alors d'occurrence du type "PRODUIT" ou plus simplement d'une entité "PRODUIT".

Les types d'entités (et aussi d'associations) ont des attributs qui les décrivent. L'ensemble des attributs d'un type d'entité doit identifier une entité parmi toutes les autres. Un sous-ensemble de ces attributs servira d'*identifiant* pour les entités (par exemple le numéro INSEE pour le type d'entité "Assuré social"). Si ce sous-ensemble n'existe pas, on pourra être amené à ajouter une rubrique uniquement pour identifier les entités (comme, par exemple, un matricule), et qui n'est pas lié aux propriétés des entités (cette rubrique ne devra pas être modifiée si les propriétés des entités sont modifiées). Une association doit être identifiée par l'ensemble des entités auxquelles elle est liée.

Nous n'irons pas plus loin dans la description théorique du modèle "entité-association".

De nombreuses méthodes de conception utilisent des schémas graphiques pour représenter le schema conceptuel des données modélisées. Ces schémas graphiques ont l'avantage de pouvoir être compris facilement par les utilisateurs non-informaticiens du système à modéliser et donc de faciliter les échanges d'informations entre ces utilisateurs et les informaticiens qui construisent l'application. La figure 3.1 est un schéma graphique très simplifié que l'on peut obtenir dans une analyse conceptuelle des données menée avec la méthode "Merise".

Ce schéma montre les types d'entités (représentés par des rectangles) et les types d'associations (représentés par des ellipses) de la base sur laquelle nous allons travailler pendant ce cours. Par exemple, la réalité décrite dans ce schéma comprend un type d'entités "EMPLOYÉ" qui correspond à l'ensemble des employés de l'entreprise modélisée. La relation "PARTICIPE À" indique qu'un employé peut participer à un "PROJET".

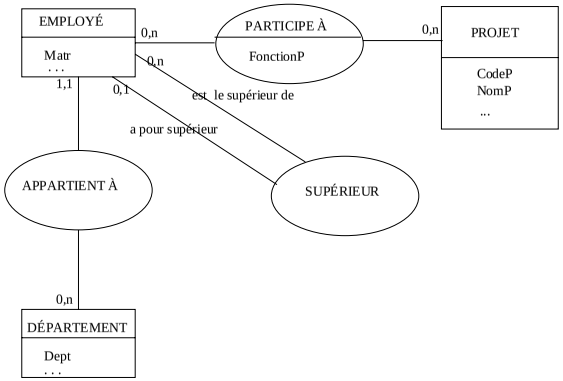

FIG. 3.1 – *Schéma conceptuel de type Merise*

3.4.2 Démarches pour obtenir un modèle conceptuel des données de type entité-association

Deux types de démarches sont possibles :

- la démarche analytique part de l'ensemble des données élémentaires (au sens des attributs définis en 4.1.1) traitées par l'organisme dont on veut modéliser le système d'information. Ces attributs sont réunis en une relation (au sens relationnel) unique que l'on va décomposer en la normalisant (voir chapitre 5),
- la démarche synthétique étudie le "discours" (au sens large de tous les documents et informations que l'on a pu recueillir) de l'utilisateur et de l'organisme pour dégager des entités et ensuite les associations qui relient ces entités.

En fait, la complexité des cas que l'on peut rencontrer dans le pratique réelle de l'analyse conceptuelle empêche d'utiliser directement la démarche analytique, surtout si on ne dispose pas d'outils informatiques d'aide à la conception.

Le plus souvent l'analyse commence avec la démarche synthétique et applique ensuite la démarche analytique sur les données liées aux entités ou associations trouvées.

3.4.3 Abandon progressif des méthodes qui s'appuient sur le modèle entité-association

Ce modèle n'est pas adapté au modèle objet et il ne convient pas vraiment à l'implantation avec les langages objets (pas de notion d'héritage par exemple). Des extensions du

modèle entité-associations ont essayé d'inclure des notions issues du modèle objet. On a ainsi vu apparaître, par exemple, des Merise "étendus" qui s'appuyaient sur un modèle entité-association étendu qui prenait en compte la notion d'héritage.

Pour ceux qui n'ont pas déjà travaillé avec les anciennes méthodes, il est plutôt conseillé d'utiliser directement les nouvelles méthodes, conçues dès le départ pour travailler avec des objets. Ces nouvelles méthodes ont gardé de nombreuses notions introduites par le modèle entité-association. On retrouve ainsi les entités, sous forme de classes et d'objets, et les associations sous forme de relations entre classes et de liens entre objets.

3.5 Représentation graphique UML et méthodologie objet

La plupart des méthodes qui s'appuient sur le modèle objet ont adopté la notation UML pour représenter les résultats des différentes étapes de l'étude.

3.5.1 Diagrammes UML

UML fournit un grand nombre de diagrammes pour représenter le système étudié, selon différents niveaux d'abstraction et selon différents points de vue.

UML part du principe que l'on peut voir une application de différents point de vue. Ces points de vue sont complémentaires et s'enrichissent mutuellement pour décrire l'application.

Voici les 9 diagrammes utilisés par UML :

des cas d'utilisation : fonctions du système et leurs interactions avec les différents acteurs (utilisateurs et autres systèmes qui interagissent avec le système étudié)

d'objets : objets et liens entre objets

de collaborations : objets et les messages qu'ils s'envoient (avec l'ordre dans lequel ils sont envoyés)

de séquence : séquencement des interactions entre les objets intervenant dans un scénario d'un cas d'utilisation (un déroulement possible d'un cas d'utilisation)

de classes : classes et relations entre classes

d'états-transitions : les différents états que peuvent prendre les objets d'une classe, avec les événements qui font passer d'un état à l'autre

d'activités : les différentes actions nécessaires à l'exécution d'une activité particulière

de déploiement : déploiement des composants de l'implémentation sur les différents dispositifs matériels (particulièrement important, par exemple, pour les applications distribuées)

de composants : utilisé pour regrouper des éléments (de tout type) en paquetages

Ces diagrammes ne sont pas indépendants, ils représentent différents aspects de l'application en cours de développement. Tout nouveau diagramme doit s'appuyer sur les diagrammes déjà conçus par l'équipe de développement. Par exemple, les diagrammes de

séquence s'appuient sur les descriptions des cas d'utilisation pour décrire un cheminement possible dans le déroulement de l'application (un *scénario*). De même, il est bien évident que les diagrammes de classes devront être compatibles avec les diagrammes d'objets.

Ce cours ne décrira pas d'une façon plus détaillées ces différents diagrammes ni les processus à suivre pour obtenir ces diagrammes. Les lecteurs intéressés par UML pourront se connecter aux sites Web
`http://www.essaim.univ-mulhouse.fr/uml/index.html` ou
`http://www.rational.com/uml/resources/quick/uml_poster.jtmpl`.

3.5.2 Démarche (très !) simplifiée pour le développement d'une application

UML n'est pas une méthode de développement. Aucune démarche particulière n'est imposée par UML pour obtenir les différents diagrammes. Chaque équipe de développement peut concevoir sa propre méthode. La société Rational qui est à l'origine d'UML, propose un processus de développement (à consulter sur son site `http://www.rational.com/uml`), centré sur les cas d'utilisation.

Cette section décrit les grandes lignes d'une démarche pour développer une application simple. La démarche exposée ci-après s'inspire de la démarche préconisée par la société Rational et conduit aux diagrammes de classe.

Pour les applications très simples, il est envisageable de concevoir directement les diagrammes de classes sans passer par des étapes préliminaires. C'est d'ailleurs ce que font implicitement les étudiants (ou d'autres...) lorsqu'ils écrivent directement une application en Java ou C++, sans analyse ou conception préalable. Dès que l'application a une certaine complexité, l'expérience a montré (quelquefois avec un coût très élevé !) que ces étapes préalables sont indispensables.

Nous montrerons au chapitre suivant comment traduire ces diagrammes de classes en un schéma relationnel pour implanter dans une base de données relationnelle les données persistantes (celles que l'on veut conserver entre deux utilisations de l'application). Dans l'état actuel du marché des SGBD, les données seront en effet le plus souvent enregistrées dans un SGBD relationnel et manipulées par le langage SQL, spécialement adapté à ce type de SGBD.

Voici donc la démarche :

- Le développement commence par la conception des diagrammes des cas d'utilisation. Cette étape indique les fonctionnalités (pas d'objets pour cette première étape) de l'application et délimite le domaine à étudier. On étudie ensuite plus ou moins séparément les différents cas d'utilisation (ils peuvent être confiés à des équipes différentes).
- Le passage à l'objet se fait en déterminant les objets qui collaboreront pour obtenir les fonctionnalités décrites par chaque cas d'utilisation. Pour cela, on suit les différents scénarios possibles des cas d'utilisation. On obtient des diagrammes d'objets et des diagrammes de séquence. Pour les cas où interviennent des algorithmes complexes, on peut être amené à utiliser des diagrammes d'états-transitions et d'activités.

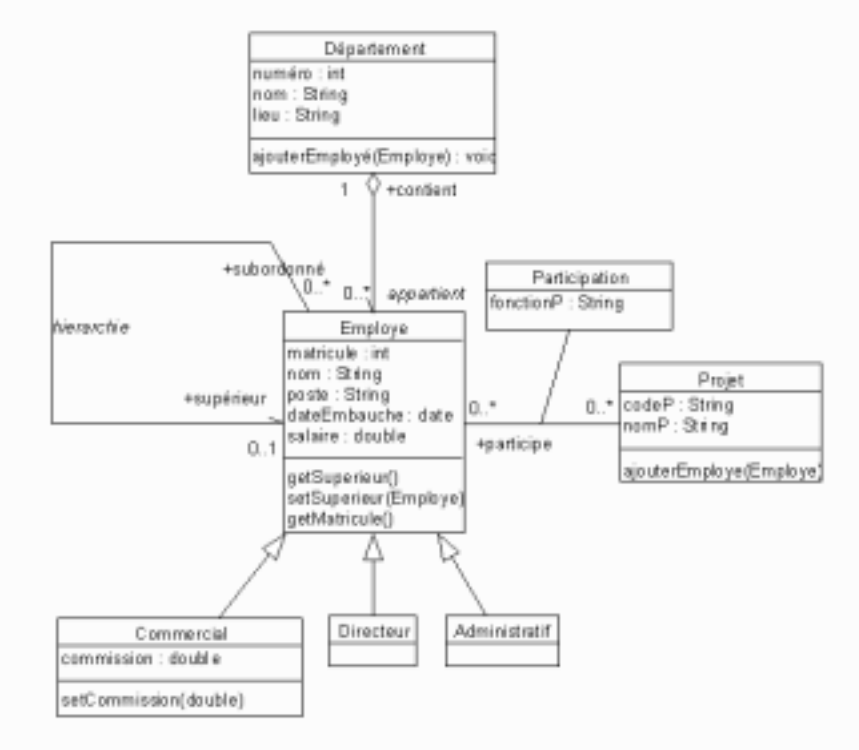

FIG. 3.2 – *Diagramme de classes UML*

– On dégage ensuite les diagrammes de classes.

3.5.3 Diagrammes de classes

La figure 3.2 est un diagramme de classes UML représentant les données qui seront enregistrées dans la base de données qui sera utilisée dans la suite de ce cours et dans le polycopié sur le langage SQL. Toutes les opérations n'ont pas été représentées.

Voici quelques notions de base.

Classes

Les classes sont représentées par des rectangles divisés en 3 sous-rectangles. Le sous-rectangle du haut contient le nom de classe, éventuellement précédé d'un *stéreotype* "précisant" le type de classe. Le seul stéréotype que nous utiliserons sera le stéréotype `<<interface>>` qui correspond à une interface au sens Java. Le nom des classes abstraites est en italique et suivi de la propriété "`{abstract}`". Les stéréotypes et propriétés peuvent être attachés à d'autres éléments UML que les classes. Les types de stéréotypes et de propriétés ne sont pas restreints par UML. Une équipe de développement peut ajouter ses propres types. Les stéréotypes permettent de classer les éléments manipulés par UML et d'étendre ainsi la sémantique du modèle UML. Ils sont placés entre des doubles signes d'inégalité. Les propriétés sont des propriétés quelconques que l'on peut attacher à un élément UML. Elles sont placées entre accolades.

Les sous-rectangles placés juste en dessous contiennent les noms et types des variables/attributs et des méthodes/opérations de la classe. Les visibilités de ces membres sont représentées par des caractères: "+" pour public, # pour protégé (*protected*), - pour privé (*private*). Les outils peuvent aussi représenter les visibilités par des petites icônes.

Les membres de classe, dits statiques (*static*), sont précédés par le signe $ ou soulignés.

Les syntaxes des déclarations des membres sont explicités dans les exemples suivants (tout est optionnel sauf le nom du membre). On remarquera que le type d'un paramètre ou le type retour d'une méthode est placé à la suite du nom et des paramètres de la méthode et précédé de ":":

```
- age : int = 20
+ ajouter(joueur : Personne, place : int) : int
```

Associations entre classes

Les objets créés par une application ne sont pas indépendants. Les liens qui existent entre les objets se traduisent par des associations entre les classes dont ils sont des instances. Dans les diagrammes de classes, ces associations sont traduites par des traits qui relient les classes. Une association a un nom. Quand ça n'est pas évident, on peut indiquer le sens de lecture de ce nom en accolant un signe > ou < au nom de l'association.

La plupart des associations sont binaires: elles associent deux classes. L'*arité* d'une association est le nombre de classes qu'elle associe. Si l'arité est supérieure à deux, l'association est représentée par un losange vers lequel arrivent les traits issus des classes qui participent à l'association.

Une classe peut être liée à une association quand l'association a des attributs. Par exemple, l'association "participe" de la figure 3.2 qui relie la classe Employé à la classe Projet, a un attribut "Fonction" qui désigne la fonction de l'employé dans le projet. Une classe est liée graphiquement au trait de l'association par un trait en pointillés.

Chaque "bout" d'une association peut avoir un rôle et une multiplicité. Le rôle précise le rôle que joueront les objets qui seront impliqués dans une tel lien. La figure 3.2 montre une association "hierarchie" avec deux rôles "subordonné" et "supérieur".

La multiplicité est très importante. À chaque bout, on indique les nombres minimums et maximums d'instances de la classe qui pourront être liés à une instance de la classe placée à l'autre bout. Voici des exemples de multiplicités :

1 : un et un seul,

0..1 : zéro ou un,

0..* : zéro ou plusieurs,

*** :** zéro ou plusieurs.

Les habitués de Merise devront se méfier : les multiplicités d'UML sont "inversées" par rapport aux multiplicités de Merise.

Par défaut, les associations sont navigables dans les deux sens : chaque classe "voit" l'autre classe. Dans certains cas une des classes n'a pas connaissance de l'autre classe. On l'indique en ajoutant une flèche au bout de cette classe et pointant vers cette classe. Par exemple, dans la figure 3.2, l'agrégation Département-Employé n'est navigable que dans le sens Département vers Employé : la classe Département voit la classe Employé mais la classe Employé ne voit pas la classe Département.

Agrégation et composition

L'agrégation est un type d'associations particulier qui traduit le fait qu'une instance d'une classe peut être liée à un certain nombre d'instances d'une autre classe. Par exemple, dans la figure 3.2, un département contient un certain nombre d'employés. Une agrégation est représentée graphiquement par un losange évidé placé au bout du trait de l'association qui est placé près de la classe "conteneur".

Dans le cas où les éléments qui sont liés au conteneur ne peuvent exister en dehors de ce conteneur, on a une composition. Dans ce cas, le losange est un losange plein et non plus évidé.

Héritage et implémentation d'interface

L'héritage est représenté graphiquement par une flèche dont le bout est un triangle évidé pointé vers la classe mère.

L'implémentation d'interface est représenté d'une façon semblable mais le trait de la flèche (dirigée vers l'interface) est en pointillés.

Deuxième partie

Modèle relationnel

Chapitre 4

Notions de base sur le modèle relationnel

4.1 Définitions

4.1.1 Relation

Une *relation* R est un sous-ensemble du produit cartésien de n ensembles D_1,D_2,...,D_n appelés *domaines*; n est appelé le *degré* de la relation. Une relation est constituée de n-uplets de la forme (d_1, d_2,..., d_n).

Plusieurs domaines peuvent être identiques. A chaque domaine est associé un nom d'*attribut* A_i qui est lié à sa signification sémantique par rapport à la relation. Ainsi les deux attributs "Salaire" et "Commission" de la relation
EMP(Matr, NomE, Poste, Datemb, Salaire, Commission)
correspondent à un même domaine que l'on pourrait appeler "Rétribution". Dans une relation, tous les attributs ont des noms différents.

Pour décrire une relation on énumère ses attributs et on donne la liste des n-uplets qui lui appartiennent (définition en *extension*). On peut aussi définir le prédicat qui permet de juger si un n-uplet appartient à la relation (on dit parfois, définition en "*intension*").

Chaque n-uplet est une *occurrence* de la relation.

On note la relation R(A_1,A_2,...,A_n).

4.1.2 Clé

Dans la théorie relationnelle, tous les n-uplets doivent être distincts (sinon on ne pourrait les distinguer en tant qu'éléments d'un ensemble). Une relation a donc au moins un sous-ensemble {A_i,A_j,...,A_k} de l'ensemble des attributs qui permet d'identifier chacun des n-uplets de la relation (l'ensemble des attributs satisfait cette condition) : deux éléments de la relation ne peuvent avoir les mêmes valeurs pour tous les attributs de ce sous-ensemble. S'il est minimal (au sens que, si on retire un des attributs, l'ensemble des attributs restants

ne permettent plus d'identifier les n-uplets de la relation), un tel sous-ensemble d'attributs est appelé une *clé candidate*.

Le concepteur choisit une de ces clés candidates comme identifiant privilégié. La clé candidate choisie s'appelle alors la *clé primaire* de la relation.

On dira qu'un attribut est *non-clé* s'il n'appartient à aucune des clés candidates.

4.1.3 Table

Dans les SGBD les relations sont représentées par des tables. Les lignes des tables correspondent aux n-uplets et les colonnes correspondent aux attributs.

DEPT

DEPT	NOMD	LIEU
10	Finances	Paris
20	Recherche	Grenoble
30	Ventes	Lyon
...	...	...

Avec les définitions des tables, les SGBD modernes permettent d'enregistrer des informations sur les clés primaires et les clés étrangères, ou même sur divers types de contraintes telles les contraintes de domaine.

4.2 Passage à un schéma relationnel

Un *schéma relationnel* est un ensemble de relations, correspondant à la structure des données manipulées par une application.

Un tel schéma comprend les informations sur les clés étrangères . Une clé étrangère est un attribut d'une relation qui fait référence à la clé primaire d'une relation.

Nous allons voir comment obtenir un tel schéma à partir d'un schéma conceptuel de type entité-association et à partir d'un diagramme de classes de type UML.

4.2.1 À partir d'un schéma "entité-association"

Pour obtenir un schéma relationnel à partir d'un schéma conceptuel de type "entité-association" (fourni par la méthode Merise), on commence par traduire chaque entité en une relation.

Par exemple, les relations correspondant aux entités du schéma conceptuel décrit dans la figure 3.1 sont (les clés primaires sont en italiques) :

EMP(*Matr*, NomE, Poste, Datemb, Salaire, Commission)

DEPT(*Dept*, NomD, Lieu)

PROJET(*CodeP*, NomP)

Les relations permettent de traduire aussi les associations grâce aux clés étrangères. Par exemple, pour représenter l'association "APPARTIENT A", il suffit d'ajouter la clé étrangère Dept dans la relation EMP :

EMP(*Matr*, NomE, Poste, Datemb, Salaire, Commission, Dept)

De même l'association SUPÉRIEUR est traduite par l'ajout de la clé étrangère Sup dans la relation EMP.

Ceci n'est possible que parce qu'un employé ne peut appartenir qu'à un seul département et n'a qu'un seul supérieur (règles de gestion de l'entreprise étudiée). Dans le cas de l'association "PARTICIPE À", il faut introduire une nouvelle relation avec deux clés étrangères qui forment la clé primaire de la relation :

PARTICIPATION(*Matr, CodeP*, FonctionP)

Le schéma relationnel devient donc :

EMP(*Matr*, NomE, Poste, Datemb, Sup, Salaire, Commission, Dept)
DEPT(*Dept*, NomD, Lieu)
PROJET(*CodeP*, NomP)
PARTICIPATION(*Matr, CodeP*, FonctionP)

Pour la traduction des associations ternaires (ou faisant intervenir plus de trois entités), on ajoute une relation dont la clé primaire est incluse dans la réunion des clés primaires des entités qui sont associées.

Il faut remarquer que l'inclusion peut être stricte. Pour cela il suffit qu'il existe une dépendance fonctionnelle (voir 5.1) d'un couple formé de deux de ces clés vers une autre clé.

4.2.2 À partir d'un diagramme de classe

Le processus est très semblable au processus décrit ci-dessus pour le schéma entité-association. Les classes jouent le rôle des entités et les relations le rôle des associations.

On commence par traduire les classes en relations.

On traduit ensuite les associations en relations. Celles qui ont un bout avec une multiplicité maximum inférieure ou égale à 1 (appelons A la classe qui est du côté de cette multiplicité) peuvent être traduite en ajoutant un attribut qui identifie une instance de la classe A dans l'autre classe. Par exemple, l'agrégation Département-Employé a une multiplicité "1" du côté de la classe Département. On peut donc traduire cette agrégation en ajoutant le numéro de département dans la relation Employé.

On doit ajouter une relation pour traduire les autres associations. Cette relation a une clé primaire composée des clés des relations représentant les classes entrant dans l'association. Les autres attributs de cette relation sont les attributs de la classe liée à cette association (si elle existe). On en a un exemple avec l'association "Participe" entre les classes Employé et Projet.

La traduction de l'héritage se fait en partageant la même clé primaire entre les deux classes. La clé primaire de la classe fille est déclarée comme une clé étrangère correspondant à la clé primaire de la classe mère. Dans le cas d'héritage multiple, la clé primaire de la classe fille est composée de clés étrangères correspondant aux clés primaires des classes mères.

4.3 Langages d'interrogation

4.3.1 Les Différents langages

Plusieurs types de langages d'interrogation des bases de données relationnelles ont été proposés. Ils s'appuient sur trois théories :

- l'algèbre relationnelle qui a inspiré le langage SQL,
- calcul relationnel des t-uples qui a inspiré le langage QUEL du SGBD "Ingres",
- calcul relationnel des domaines qui a inspiré le langage QBE (*Query By Example*) d'IBM.

On a pu démontrer qu'ils étaient équivalents dans le sens où ils permettent de désigner les mêmes ensembles de données.

4.3.2 Calcul relationnel

Le calcul relationnel des t-uples et le calcul relationnel des domaines reposent sur le calcul des prédicats du premier ordre. Cette théorie mathématique étudie les formules logiques construites avec un ensemble de prédicats (c'est-à-dire de propositions qui peuvent être vraies ou fausses dans un certain contexte), les opérateurs "et", "ou", "négation", "implication logique", et qui peuvent comporter des variables et les opérateurs "$\forall$" et "$\exists$".

À chaque formule logique correspond l'ensemble des données qui vérifient cette formule. L'interrogation de la base de données consiste donc à énoncer une formule qui correspond aux données que l'on souhaite extraire de la base.

La différence essentielle entre le calcul relationnel des t-uples et le calcul relationnel des domaines est l'ensemble dans lequel les variables prennent leurs valeurs. Pour le calcul relationnel des t-uples il s'agit de l'ensemble des t-uples de la base. Pour le calcul relationnel des domaines chaque variable prend ses valeurs dans un domaine des attributs de la base.

Pour les prédicats, on se limite à n'utiliser que les opérateurs qui comparent des valeurs d'attributs en utilisant les opérateurs habituels ($>, \geq, <, \leq, \neq$). Pour le calcul relationnel des t-uples on ajoute l'opérateur unaire qui indique qu'une variable prend ses valeurs dans une relation (par exemple, EMP(V) indique que la variable V appartient à la relation EMP). Pour le calcul relationnel des domaines, on ajoute l'opérateur unaire qui indique qu'une variable prend ses valeurs dans un domaine (par exemple, EMP(Matr:v) indique que la variable v prend ses valeurs dans le domaine de l'attribut Matr de la relation EMP).

Exemples 4.1

(a) Noms des départements dans lesquels travaille Martin :

- en calcul relationnel des t-uples :
  {D.Nomd / DEPT(D) *et* $\exists$ E EMP(E) *et* (E.NomE='MARTIN') *et* (E.Dept = D.Dept)}
- en calcul relationnel des domaines :
  {d / $\exists$ n,x DEPT(Dept:x, Nomd:d) *et* EMP(NomE:n, Dept:x) *et* (n='MARTIN')}

(b) Noms des employés qui participent à un projet :

- en calcul relationnel des t-uples :
  {E.NomE / EMP(E) *et* ($\exists$ P PARTICIPATION(P) *et* E.Matr=P.Matr)}
- en calcul relationnel des domaines :
  {n / $\exists$ m EMP(Matr:m, NomE:n) *et* PARTICIPATION(Matr:m)}

4.4 Algèbre relationnelle

Nous nous étendrons davantage sur l'algèbre relationnelle qui a inspiré le langage SQL qui va faire l'objet de la deuxième partie du cours.

Le principe de l'algèbre relationnelle est différent du calcul relationnel. On se donne des opérateurs que l'on applique à des relations de la base de données. Les opérateurs sont de deux types : relationnels et ensemblistes.

4.4.1 Opérateurs relationnels

Projection

La projection d'une relation est une relation qui n'a que certains attributs de la première relation.

La projection de la relation R sur les attributs A_1,... ,A_n sera notée **R[A_1,...,A_n]** dans ce cours (il n'y a pas de standard bien défini pour ces notations).

Exemple : DEPT[Dept, NomD]

Sélection (ou restriction)

La sélection d'une relation est une relation qui n'a que les n-uplets de la première relation qui vérifient une condition c.

La sélection de la relation R par la condition c sera notée **R/c**.

Exemple : DEPT / Dept > 20

Jointure

Nous donnerons d'abord la définition de l'*équi-jointure* qui est la plus fréquemment employée :

Soient deux relations R et S, A un attribut de R, B un attribut de S, qui ont le même domaine de valeurs. La jointure de R sur A et de S sur B est la relation dont les n-uples sont obtenus par concaténation des n-uples de R et des n-uples de S qui ont la même valeur pour les attributs A et B.

Cette jointure sera notée **R J{A=B} S**.

Souvent on supprime l'attribut B de l'équi-jointure (puisque les valeurs de A et de B sont égales) et on obtient alors la *jointure naturelle* de R et de S.

Il arrive souvent que les deux attributs (sur lesquels s'opère la jointure) ont le même nom. Dans ce cas, la jointure naturelle sera notée **R J{A} S**.

Exemple 4.2

La jointure naturelle de la relation EMP et de la relation DEPT sur les deux attributs DEPT est représentée par la troisième table ci-dessous :

EMP

Matr	NomE	...	Dept
1050	DURAND	...	10
832	DUPOND	...	20
900	DUVAL	...	10

DEPT

Dept	NomD	Lieu
10	Finances	Paris
20	Recherche	Grenoble
30	Ventes	Lyon
...	...	...

EMP J{Dept} DEPT

Matr	NomE	...	Dept	NomD	Lieu
1050	DURAND	...	10	Finances	Paris
900	DUVAL	...	10	Finances	Paris
832	DUPOND	...	20	Recherche	Grenoble

Plus généralement, on peut joindre deux relations avec une condition de comparaison différente de l'égalité : $>, \geq, <, \leq, \neq$.

Par exemple, on peut joindre les n-uplets de la relation EMP avec tous les n-uplets qui ont un numéro de matricule (MATR) inférieur. On joint la relation EMP avec elle-même et le lien concerne les deux attributs MATR avec l'opérateur de comparaison "<" et non plus l'égalité.

Ce genre de jointure est beaucoup moins utilisé que l'équi-jointure.

Elle sera notée **R J{A *c* B} S** (où c est un des opérateurs de comparaison $>, \geq, <, \leq, \neq$). Par exemple, EMP J{Salaire < Salaire} EMP, si l'on veut, pour chaque employé, une liste des employés qui ont un salaire plus élevé.

Division

Soient deux relations R(A,B) et S(C), où A, B et C représentent des ensembles d'attributs tels que B et C ont des domaines compatibles.

La division de R par S sur l'attribut B est la relation D définie par :

$$D = \{a \in R[A] / \forall b \in S, (a,b) \in R\} = \{a \in R[A] / \ \nexists b \in S, (a,b) \notin R\}$$

On notera D = **R** $\div_B$ **S**.

Il arrive souvent que les attributs B et C aient les mêmes noms. Dans ce cas, on note la division plus simplement par : **R** $\div$ **S**.

Pratiquement, D s'obtient en prenant toutes les valeurs de l'attribut A de R qui sont associées à toutes les valeurs de S.

Cette opération s'appelle la division par analogie avec la division euclidienne de deux nombres entiers. En effet, R $\div$ S est le plus grand sous-ensemble D de R[A] tel que D $\times$ S $\subset$ R, de même que a $\div$ b (division entière d'un nombre entier a par un nombre entier b) est le plus grand entier d tel que d $\times$ b $\leq$ a.

Exemple 4.3

Si R = {(x,1), (y,2), (x,3), (z,1)} et S = {1,3}, la division de R par S sur le premier attribut de R et l'attribut de S est D = {x} car x est associé à 1 et à 3 dans la relation R.

R

A	B
x	1
y	2
x	3
z	1

S

C
1
3

D

A
x

La division fournit la réponse au type de question suivant : quels sont les "A" qui sont associés à tous les "C" (le mot "tous" est caractéristique de ce genre de question).

Exemple 4.4

Quels sont les départements qui participent à tous les projets?

La réponse à cette question est fournie par R $\div$ S où

R = (PARTICIPATION J{Matr} EMP)[Dept, CodeP] et S = PROJET[CodeP]

4.4.2 Opérateurs ensemblistes

Si deux relations ont des attributs qui prennent leurs valeurs dans des domaines identiques (ou compatibles), on obtient une troisième relation en prenant la réunion **R1** $\cup$ **R2**, l'intersection notée **R1** $\cap$ **R2**, la différence des deux relations (en tant qu'ensembles de t-uples) notée **R1** $-$ **R2**. On peut aussi effectuer le produit cartésien de deux relations quelconques notée **R1** $\times$ **R2**.

Chapitre 5

Normalisation d'un schéma relationnel

Si l'on n'y prend garde, une mauvaise répartition des données dans les relations peut occasionner de graves problèmes lors de l'évolution de la base.

La normalisation des relations permet d'éviter ces problèmes. L'idée essentielle de la normalisation est : autant que possible "un seul fait dans un seul lieu" c'est-à-dire une seule notion sémantique par relation. Pour obtenir cela, le processus de normalisation conduit le plus souvent à éclater des relations en plusieurs relations. Il existe plusieurs degrés dans les formes normales. Les différentes formes normales sont étudiées en 5.3.

La notion de forme normale repose sur les notions de clé (notion étudiée en 4.1.2) et de dépendance fonctionnelle.

5.1 Dépendances fonctionnelles

Il y a une *dépendance fonctionnelle* entre deux attributs X et Y d'une même relation R (on dit aussi que Y dépend de X) si toute valeur pour l'attribut X détermine une valeur bien définie pour l'attribut Y. On note $\mathbf{X} \longrightarrow \mathbf{Y}$. Les attributs X et Y peuvent être composés de plusieurs rubriques.

Plus formellement, soient X et Y deux ensembles de rubriques de la relation R. $\mathbf{X} \longrightarrow \mathbf{Y}$ si et seulement si
E1 et E1 sont deux n-uplets de la relation R qui ont les mêmes valeurs pour les rubriques de X, alors ils ont nécessairement les mêmes valeurs pour les rubriques de Y.

Exemples 5.1

(a) Il y a dépendance fonctionnelle d'une clé primaire vers tous les autres attributs d'une relation.

(b) Dans la relation (Nom, Prénoms, Ville, Code postal, Département),
on a la dépendance fonctionnelle Code postal $\longrightarrow$ Département
(mais pas Ville $\longrightarrow$ Département, car 2 villes peuvent avoir le même nom)

(c) Dans une classe d'étudiants sans homonymes (pour le nom et le prénom), dans la relation (Nom, Prénom, Moyenne, Âge, Enseignant),

on a les dépendances fonctionnelles
(Nom, Prénom) $\longrightarrow$ Moyenne
(Nom, Prénom) $\longrightarrow$ Âge

5.2 Schéma relationnel normalisé

Un schéma relationnel est normalisé pour une certaine forme normale si les relations qui le composent sont toutes dans la forme normale spécifiée.

Normaliser un schéma relationnel c'est le transformer en un schéma relationnel normalisé *équivalent*, c'est-à-dire, contenant les mêmes informations que le schéma de départ. En général cela consiste à éclater les relations non normalisées en plusieurs relations normalisées qui redonnent les relations d'origine par jointure. La décomposition doit être sans perte de données et autant que possible sans perte de dépendance (voir 5.4).

L'idée "un seul fait dans un seul lieu" se traduit souvent par la suppression des dépendances fonctionnelles intra-relation autres que celles liées aux clés candidates.

Exemple 5.2

La dépendance
Dept $\longrightarrow$ NomD
dans la relation
(*Matr*, NomE, Dept, NomD)
va occasionner des anomalies lors des modifications des données :

- anomalies d'insertion : on ne peut ajouter de nouveau département (numéro et nom) si le département ne comporte pas d'employé,
- anomalies de modification : si l'on change le nom d'un département, on doit le changer dans toutes les lignes des employés de ce département,
- anomalies de suppression : si on supprime le dernier employé d'un département, on perd l'information numéro-nom du département.

La normalisation de la relation (*Matr*, NomE, Dept, Nomd) par sa décomposition en deux relations
(*Matr*, NomE, Dept)
(*Dept*, Nomd)
aurait permis d'éviter les problèmes pendant la mise à jour des données.

5.3 Formes normales

Les formes normales les plus utilisées sont au nombre de 6 : les formes normales numérotées de 1 à 5 et la forme normale de Boyce-Codd. Elles représentent un certain degré de

normalisation des relations. Nous étudierons dans ce cours les 3 premières formes normales et la forme normale de Boyce-Codd.

Remarque 5.1

Pour tous les exemples et contre-exemples cités ici, on supposera l'existence des dépendances fonctionnelles "naturelles" que l'on rencontreraient dans les cas concrets. Par exemple, on suppose l'existence de la dépendance fonctionnelle
CodeLivre $\longrightarrow$ Titre

$1^{ère}$ forme normale : une relation est en première forme normale si elle ne comporte pas d'attribut multivalué. La représentation des données sous forme de relation implique cette première forme normale.

Exemple de pseudo-relation qui n'est pas en $1^{ère}$ forme normale :
"LIVRE(CodeLivre, Titre, Auteurs)"
n'est pas en $1^{ère}$ forme normale car un livre peut avoir plusieurs auteurs.

On peut décomposer cette relation en deux relations qui seront en $1^{ère}$ forme normale :
LIVRE(*CodeLivre*, Titre)
AUTEURS(*CodeLivre*, *Auteur*)

$2^{ème}$ forme normale : une relation est en deuxième forme normale si la relation est en première forme normale et si chaque attribut non-clé (c'est-à-dire qui n'appartient pas à une clé candidate) dépend *pleinement* des clés candidates.

Une dépendance fonctionnelle d'un attribut par rapport à un ensemble d'attributs est *pleine* si l'attribut ne dépend pas d'un sous-ensemble strict de cet ensemble d'attributs. On dit aussi que la dépendance fonctionnelle est *élémentaire*. Pour qu'une dépendance fonctionnelle soit non pleine, il est nécessaire que la source de la dépendance soit composée de plusieurs rubriques. On doit avoir par exemple, (R1, R2) $\longrightarrow$ R et R1 $\longrightarrow$ R.

Exemple de relation qui n'est pas en $2^{ème}$ forme normale :
PARTICIPATION(*Matr*, *CodeP*, NomE, FonctionP)
n'est pas en $2^{ème}$ forme normale car le nom d'un employé ne dépend que de son matricule et pas du projet auquel il appartient.

Problèmes occasionnés lors de la mise à jour des données : si on modifie le nom d'un employé, on doit le modifier dans toutes les lignes qui correspondent aux divers projets on ne peut insérer un nouvel employé s'il ne participe pas à un projet on risque de supprimer les renseignements sur un employé s'il ne participe plus à aucun projet.

On peut décomposer cette relation en deux relations qui seront en $2^{ème}$ forme normale :
PARTICIPATION(*Matr*, *CodeP*, FonctionP)
EMP(*Matr*, NomE)

$3^{ème}$ forme normale : une relation est en troisième forme normale si tout attribut non-clé (n'appartenant pas à une clé candidate) ne peut dépendre que d'une clé candidate.

Dans le cas où il n'y a qu'une seule clé candidate, cette définition est équivalente à la suivante : la relation est en deuxième forme normale et il y a dépendance *directe* (non transitive, c'est-à-dire une dépendance qui pourrait être retrouvée par composition de deux dépendances fonctionnelle) des attributs qui n'appartiennent pas à la clé par rapport à la

clé primaire.

Remarque 5.2

Les dépendances fonctionnelles vers une rubrique qui appartient à une clé sont autorisées. C'est ce qui fait la différence avec la forme normale de Boyce-Codd définie ci-dessous.

Exemple de relation qui n'est pas en $3^{ème}$ forme normale :
EMP(*Matr*, NomE, Dept, NomD)
n'est pas en $3^{ème}$ forme normale car le nom du département dépend du numéro du département.

Comme pour la $2^{ème}$ forme normale, on rencontrera des anomalies lors des mises à jour de relations qui ne sont pas en $3^{ème}$ forme normale. On sera par exemple obligé de dupliquer les renseignements concernant les départements.

On peut décomposer cette relation en deux relations qui seront en $2^{ème}$ forme normale :
EMP(*Matr*, NomE, Dept)
DEPT(*Dept*, NomD)

Forme normale de Boyce-Codd : une relation est en forme normale de Boyce-Codd si les seules sources de dépendances fonctionnelles sont les clés candidates. Cette forme normale est souvent appelée BCNF (d'après les initiales en anglais).

Ceci implique que la relation est en troisième forme normale mais une relation peut être en troisième forme normale sans être dans la forme normale de Boyce-Codd. Cela ne peut arriver que si deux clés candidates se "chevauchent" (ont en commun une ou plusieurs rubriques) et qu'une partie d'une des clés candidates dépend d'un attribut qui appartient à l'autre clé candidate.

L'avantage de travailler avec des relations en forme normale de Boyce-Codd est que les seules dépendances fonctionnelles intra-relation à respecter lors des ajouts ou modifications de données de la base sont celles qui sont liées à l'unicité des clés d'une relation (voir 5.4.2).

Exemples 5.3

(a) Si on suppose que le nom d'un employé peut être une clé candidate, la relation
EMP_PROJET(*Matr*, *CodeP*, NomE, FonctionP)
est en $3^{ème}$ forme normale. En effet les 2 clés candidates sont
(Matr, CodeP) et (NomE, CodeP)
et le seul attribut qui n'appartient pas à une clé candidate est FonctionP qui ne dépend pas d'un sous-ensemble strict d'une des clés candidates.
Cette relation n'est pas en BCNF car
Matr $\longrightarrow$ NomE
est une dépendance fonctionnelle dont la source n'est pas une clé candidate. Elle va occasionner des redondances et des problèmes de mises à jour. Elle peut être décomposées dans les 2 relations suivantes qui sont en BCNF :
EMP(*Matr*, NomE) PROJET(*Matr*, *CodeP*, FonctionP)

(b) Un cas plus intéressant correspond au schéma suivant d'une relation :
(*clé1, clé2*, rubrique3)

avec les dépendances fonctionnelles
(clé1, clé2) $\longrightarrow$ rubrique3
rubrique3 $\longrightarrow$ clé2
est en troisième forme normale mais pas en forme normale de Boyce-Codd (voir remarque ci-dessus sur la troisième forme normale). On verra un exemple de cette situation dans la section 5.4.2 (décomposition avec perte de dépendance).

4ème et 5ème formes normales : elles sont moins utilisées et elles ne seront pas étudiées dans ce cours. Elles permettent d'éviter certaines redondances dans le cas de dépendances autres que les dépendances fonctionnelles. Elles correspondent à des conditions plus strictes que les formes normales étudiées ci-dessus : si une relation est en cinquième forme normale, elle est en quatrième forme normale et si elle est en quatrième forme normale, elle est en troisième forme normale.

5.4 Pertes de données et de dépendances

Les pertes de données ou de dépendances peuvent survenir lors des décompositions des relations.

5.4.1 Perte de données

Une décomposition en plusieurs relations est dite "sans perte de données" si la jointure naturelle (voir 4.4.1) des relations obtenues dans la décomposition redonne la relation de départ.

Voici un exemple de décomposition avec perte de données :

Soit la relation
EMP_DEPT(*Matr*, *Dept*, NomE, Salaire, NomD)
avec les dépendances fonctionnelles Matr $\longrightarrow$ Nome et Dept $\longrightarrow$ NomD. Cette relation ne peut être décomposé en
EMP(*Matr*, NomE, Salaire)
DEPT(*Dept*, NomD, Salaire)
car la jointure naturelle de ces deux relations sur l'attribut Salaire donnera plus de n-uplets que la relation de départ si plusieurs employés ont le même salaire (la "perte" de données est en fait un surplus de données parasites). Ceci est dû au fait que la rubrique commune (Salaire) n'est pas source d'une dépendance fonctionnelle vers les autres rubriques d'une des deux relations de la décomposition.

Théorème important : soit une relation R(A, B, C) où A, B, C sont des ensembles d'attributs disjoints avec la dépendance fonctionnelle B $\longrightarrow$ C.

On peut démontrer que la décomposition de R en les relations R1(A $\cup$ B) et R2(B $\cup$ C) est sans perte de données.

Comparez avec la décomposition ci-dessus avec perte de dépendance avec la suivante :
EMP(*Matr*, NomE, Salaire, Dept)
DEPT(*Dept*, NomD)

Ici l'attribut commun est la source de la dépendance fonctionnelle Dept $\longrightarrow$ NomD et on n'a donc pas de perte de données.

5.4.2 Perte de dépendances

Soit une relation R qui est décomposée en deux relations R1 et R2. La projection de l'ensemble des dépendances de R sur la relation R1 s'obtient en ne gardant des dépendances de départ de R que celles dont les attributs (de la source et du but) sont tous des attributs de la relation R1.

Une décomposition en plusieurs relations est dite "sans perte de dépendances" si on peut retrouver logiquement (par exemple par transitivité) les dépendances de départ à partir de la projection de l'ensemble des dépendances de départ sur les relations de la décomposition.

Si on décompose une relation en plusieurs relations avec perte de dépendance, on ne peut travailler indépendamment dans chacune des relations de la décomposition sans s'occuper des autres relations. Lors des mises à jour, on ne peut en effet s'assurer que les dépendances "perdues" sont vérifiées qu'en consultant les autres relations, ce qui peut occasionner des traitements coûteux.

Lorsque l'on a le choix entre plusieurs décompositions il faut évidemment préférer celles qui préservent les dépendances. Il est cependant des cas où il faut choisir entre une perte de dépendance ou une perte de normalisation comme dans l'exemple suivant :

Supposons que l'on veuille intégrer la règle de gestion "Deux personnes d'un même département ne peuvent participer à un même projet." au schéma relationnel des données donné à la page 24.

Cette règle de gestion implique la dépendance fonctionnelle
(Dept, CodeP) $\longrightarrow$ Matr

Si l'on veut tenir compte de la nouvelle règle de gestion, on obtient le nouveau schéma relationnel :
EMP(*Matr*, NomE, Poste, Datemb, Sup, Salaire, Commission, Dept)
DEPT(*Dept*, NomD, Lieu)
PROJET(*CodeP*, NomP)
PARTICIPATION2(*Dept, CodeP*, Matr, FonctionP)

Remarque 5.3
On ne peut enlever l'attribut Dept de la relation EMP malgré son ajout dans la relation PARTICIPATION2 car on perdrait le département des employés qui ne participent à aucun projet.

La relation PARTICIPATION2 est en troisième forme normale mais elle n'est pas en forme normale de Boyce-Codd à cause de la dépendance fonctionnelle Matr $\longrightarrow$ Dept.

Si on veut un schéma relationnel avec des relations en forme normale de Boyce-Codd, on peut penser à décomposer la relation PARTICIPATION2 et on sera ramené au schéma de la page 24 : PARTICIPATION2 va s'éclater en
R1(*CodeP, Matr*, FonctionP)
R2(*Matr*, Dept).

R2 est redondante avec EMP et R1 n'est autre que la relation PARTICIPATION du schéma de la page 24.

Cette décomposition est sans perte de données comme on peut le vérifier aisément (il est évident que l'on retrouve exactement la relation PARTICIPATION2 à partir de R1 et R2 grâce à la dépendance fonctionnelle Matr $\longrightarrow$ Dept) mais pas sans perte de dépendance.

En effet, la projection des dépendances ne contient que les dépendances Matr $\longrightarrow$ Dept et (CodeP, Matr) $\longrightarrow$ FonctionP. On ne peut en déduire la dépendance
(Dept, CodeP) $\longrightarrow$ Matr.

Cette perte de dépendance fait que l'on ne peut vérifier la contrainte d'intégrité "Deux personnes d'un même département ne peuvent participer à un même projet" en travaillant seulement sur une des relations. On est obligé de refaire une jointure (même partielle) des deux tables PARTICIPATION2 et EMP pour vérifier la contrainte d'intégrité, par exemple, si on veut changer quelqu'un de projet ou ajouter un nouvel employé.

Il faut donc faire un choix entre la perte de dépendance si l'on éclate la relation en deux relations en forme normale de Boyce-Codd et les problèmes éventuels dus au fait que la relation PARTICIPATION2 n'est pas en forme normale de Boyce-Codd. Par exemple, si on travaille avec PARTICIPATION2, et si un employé qui participe à plusieurs projets change de département, il faudra changer le département dans toutes les lignes de PARTICIPATION2 qui concernent cet employé.

La relation
PARTICIPATION2(*Dept, CodeP*, Matr, Fonction)
est un exemple de relation "atomique" qui ne peut être décomposée sans perte de dépendance.

On peut démontrer que l'on peut toujours normaliser un schéma en troisième forme normale sans perte de données ni perte de dépendances. Il n'est pas toujours possible de conserver les dépendances si l'on veut normaliser un schéma avec uniquement des relations en quatrième ou cinquième formes normales ou des formes normales de Boyce-Codd.

Troisième partie

Utilisation et implantation des SGBD relationnels

Chapitre 6

Utilisation d'un SGBD relationnel

Cette section rassemble, sans entrer dans les détails, quelques notions utiles pour l'utilisation d'un SGBD relationnel. Le cours sur le langage SQL donne quelques applications pratiques de ces notions.

6.1 Intégrité des données

6.1.1 Transaction

Une transaction est un ensemble de modifications de la base qui forment un tout indivisible : il faut effectuer ces modifications entièrement ou pas du tout, sous peine de laisser la base dans un état incohérent.

Exemple 6.1
Une transaction peut transférer une somme d'argent entre deux comptes d'un client d'une banque. Elle comporte deux ordres : un débit sur un compte et un crédit sur un autre compte. Si un problème empêche le crédit, le débit doit être annulé.

Une transaction est terminée :

- soit par une validation qui entérine les modifications,
- soit par une annulation qui remet la base dans son état initial.

Deux transactions ne peuvent imbriquées : dans une session de travail, la transaction en cours doit se terminer avant qu'une nouvelle transaction ne puisse démarrer.

Une transaction commence au début d'une session de travail ou juste après la fin d'une transaction précédente. L'utilisateur peut à tout moment valider la transaction en cours (commande COMMIT en SQL). Les modifications deviennent alors définitives et visibles à tous les utilisateurs.

Tant que la transaction n'est pas validée, les insertions, modifications et suppressions qu'elle contient n'apparaissent qu'à l'utilisateur qui est en train de l'exécuter. Tous les autres utilisateurs voient la base dans l'état où elle était avant la transaction.

L'utilisateur peut annuler la transaction en cours (commande ROLLBACK en SQL). Toutes les modifications depuis le début de la transaction sont annulées.

Ce mécanisme est aussi utilisé pour assurer l'intégrité de la base en cas de fin anormale d'une tâche utilisateur : il y a automatiquement annulation des transactions non terminées.

6.1.2 Accès concurrents

Problèmes liés aux accès concurrents

Il peut arriver que plusieurs processus veuillent modifier en même temps les mêmes données. Dans ce cas, il peut se produire des pertes de données si l'on ne prend pas certaines précautions.

Étudions quelques cas d'école des problèmes liés aux accès concurrents.

Mises à jour perdues Si deux processus mettent à jour en même temps la somme S contenue dans un compte client d'une banque en enregistrant deux versements de 1000 et de 2000 francs, on peut avoir

Temps	Processus P1	Processus P2
t1	Lire S	
t2		Lire S
t3	S = S + 1000	
t4		S = S + 2000
t5	Enregistrer S	
t6		Enregistrer S

Finalement, la valeur du compte sera augmentée de 2000 francs au lieu de 3000 francs.

Pour éviter ce genre de situation, les SGBD bloquent l'accès aux tables (ou parties de tables : blocs mémoire ou lignes par exemple) lorsque de nouveaux accès pourraient occasionner des problèmes. Les processus qui veulent accéder aux tables bloquées sont mis en attente jusqu'à ce que les tables soient débloquées. Cependant ceci peut aussi amener à l'interblocage de processus (*deadlock* en anglais) comme dans l'exemple suivant :

Temps	Processus P1	Processus P2
t1	Verrouiller la table A	
t2		Verrouiller la table B
t3	Verrouiller la table B	
t4	*Attente*	Verrouiller la table A
t5		*Attente*

On remarquera que ce type d'interblocage ne peut arriver si tous les utilisateurs bloquent les tables dans le même ordre.

Lecture inconsistante ou lecture impropre Une transaction t2 lit une valeur V donnée par une autre transaction t1. Ensuite la transaction t1 annule son affectation de V et la valeur lue par t2 est donc fausse.

Temps	Processus P1	Processus P2
t1	V = 100	
t2		Lire V
t3	ROLLBACK	

Ce cas ne peut arriver si les modifications effectuées par une transaction ne sont visibles par les autres qu'après validation (Commit) de la transaction. C'est le plus souvent le cas.

Lecture non répétitive, ou non reproductible, ou incohérente Une transaction lit deux fois une même valeur et ne trouve pas la même valeur les deux fois.

Temps	Processus P1	Processus P2
t1	Lire V	
t2		V = V + 100
t3		COMMIT
t4	Lire V	

Pour éviter ceci, P1 devra bloquer les données qu'il veut modifier entre les temps t1 et t3, pour empêcher les autres transactions de les modifier (voir, par exemple, les blocages explicites d'Oracle dans le cours sur le langage SQL).

Lignes fantômes Une sélection de t-uplets récupère des lignes qui sont modifiées par une autre transaction et ne vérifient plus le critère de la sélection. La même sélection relancée une deuxième fois ne retrouve donc plus ces lignes.

Temps	Processus P1	Processus P2
t1	Sélectionne les lignes avec V = 10	
t2		V = 20 pour toutes les lignes
t3	Sélectionne les lignes avec V = 10	

Là encore la solution est le blocage explicite des lignes en question (voir cours SQL).

Traitement des accès concurrents par les SGBD

Les SGBD gèrent automatiquement les accès concurrents de plusieurs utilisateurs sur les mêmes lignes des tables. Avec Oracle, si un utilisateur est en train de modifier des lignes d'une table, les autres utilisateurs peuvent lire les données telles qu'elles étaient avant ces dernières modifications (jamais de temps d'attente pour la lecture), mais les autres utilisateurs sont bloqués automatiquement s'ils veulent modifier ces mêmes lignes (pour éviter les risques de pertes de données signalés plus haut).

De plus, Oracle assure une "lecture consistante" des données pendant l'exécution d'un ordre SQL ; par exemple, un ordre SELECT ou UPDATE va travailler sur les lignes telles qu'elles étaient au moment du début de l'exécution de la commande, même si entre-temps un autre utilisateur a confirmé (COMMIT) des modifications sur certaines de ces lignes.

L'utilisateur (ou le programmeur) peut ne pas se satisfaire du blocage effectué automatiquement par Oracle. On verra dans le cours sur le langage SQL qu'il peut bloquer des tables ou des lignes à sa convenance.

La gestion des transactions doit, si possible, rendre les transactions "*sérialisable*" : l'exécution simultanée d'un ensemble de transactions doit donner le même résultat qu'une exécutionséquentielle de ces transactions. Dans le cas du SGBD Oracle, une option permet d'imposer la "sérialisabilite" (au détriment des performances). Sinon, la cohérence de l'état de la base est sous la responsabilité des utilisateurs et des programmeurs.

6.1.3 Contraintes d'intégrité

Des contraintes d'intégrité peuvent être entrées dans la définition d'une base ; par exemple :

- définition de la clé primaire d'une table,
- clé étrangère dont les valeurs doivent exister dans une autre table,
- unicité des valeurs d'une rubrique,
- fourchette pour les valeurs acceptables.

Dans ce cas, les commandes qui créent des lignes qui ne respectent pas ces contraintes sont automatiquement annulées. Plus précisément, si une seule des lignes ne respecte pas une contrainte d'intégrité, toutes les autres modifications éventuelles de la commande sont annulées.

6.2 Sécurité

6.2.1 Contrôle de l'accès à la base

L'accès à une base de données nécessite un nom d'utilisateur et un mot de passe associé.

Chaque utilisateur a des *privilèges* d'accès à la base. Ces privilèges correspondent à certains droits, par exemple, droit de travailler avec les tables existantes, de créer de nouvelles tables, de créer de nouveaux utilisateurs.

6.2.2 Contrôle de l'accès aux données

Les données d'une table appartiennent au créateur de la table. Celui-ci peut donner aux autres utilisateurs des droits sur ses tables, par exemple, droit de consulter, d'ajouter, de supprimer des données.

Des restrictions d'accès aux données peuvent aussi passer par l'utilisation de vues. Une vue est une table virtuelle qui est constituée de colonnes qui appartiennent à des tables déjà créées. On peut donner des droits d'accès à des vues sans donner de droits d'accès aux tables sous-jacentes. On peut ainsi donner des droits sur seulement certaines lignes et certaines colonnes d'une ou de plusieurs tables.

6.3 Résistance aux pannes

6.3.1 Sauvegardes

Les SGBD offrent des outils et des mécanismes automatiques pour réparer les dommages créés par les pannes.

Pour en profiter, l'administrateur doit faire régulièrement des sauvegardes de la base et des fichiers indispensables à son bon fonctionnement.

Le SGBD enregistre automatiquement toutes les instructions exécutées sur la base dans des fichiers appelés fichiers "*log*" (fichiers "Redo Log" de Oracle). Ces fichiers sont utilisés durant les redémarrages du SGBD après les pannes.

L'administrateur de la base peut archiver ces fichiers "log" sur des supports magnétiques. Le fonctionnement est légèrement ralenti mais c'est une assurance contre des pertes trop importantes après panne.

6.3.2 Types de pannes

Les pannes peuvent être d'origine logicielle (SGBD ou applications qui accèdent aux données) ou matérielle (le plus souvent des problèmes dus aux disques où sont enregistrés les données). Pour les bases de données réparties les réseaux peuvent aussi être la cause de pannes.

Le traitement des pannes est différent si la panne a seulement occasionné une interruption du logiciel en laissant la base dans un état cohérent, ou si la base a été endommagée et ne peut plus être utilisée.

6.3.3 Reprise après incident

Cas où la panne n'a pas endommagé les fichiers de la base

Dans ce cas, lorsque le SGBD redémarre après la panne, il commence par relancer l'exécution des instructions confirmées par un COMMIT et dont les modifications n'ont pas été enregistrées dans la base (par exemple en raison de l'écriture asynchrone des modification des données dont on parle en 7.3). Il utilise pour cela les fichiers "log" enregistrés automatiquement pendant le fonctionnement du SGBD (voir 7.4). Ensuite les transactions qui ont été annulées par un ROLLBACK, ou qui n'ont pas été confirmées par un COMMIT avant la panne, sont annulées.

Cas où la panne a endommagé les fichiers de la base

Dans ce cas, l'administrateur de la base doit recharger la dernière sauvegarde. Si l'administrateur a choisi d'archiver toutes les modifications de la base, il peut relancer toutes les instructions exécutées jusqu'à la panne ou au moins jusqu'au dernier archivage intact. Sinon, il faudra relancer à la main toutes les transactions exécutées depuis cette dernière sauvegarde.

Ces mécanismes sont d'autant plus souvent utiles que, pour des raisons de performance, les modifications apportées à la base ne sont pas immédiatement enregistrées dans la base après la confirmation des transactions par une instruction COMMIT.

6.4 Dictionnaire des données

Toutes les définitions utilisées par le SGBD sont enregistrées dans la base de données : tables, vues, utilisateurs, index, contraintes d'intégrité, etc. L'ensemble des tables ou vues où sont enregistrées ces définitions est appelé le dictionnaire des données.

Certaines tables et vues sont accessibles en lecture par tous les utilisateurs, ou du moins pour les définitions des objets dont ils sont propriétaires. D'autres sont réservées aux administrateurs de la base.

6.5 Performances

Bien que les langages d'utilisation des SGBD relationnels sont des langages qui ne font pas référence au cheminement que l'on doit suivre pour accéder aux données, cet accès peut être accéléré par la création d'index ou par une organisation physique particulière des données de la base.

On peut créer un index sur une rubrique (ou plusieurs rubriques concaténées) pour accélerer l'accès aux lignes qui ont une certaine valeur pour cette rubrique ou le tri des lignes dans l'ordre de rangement de l'index. À la différence des SGBD des générations précédentes, l'existence ou non d'un index ne joue en rien sur la possibilité d'obtenir des données. Elle joue seulement sur la rapidité de recherche des données.

D'autres structures permettent d'améliorer les performances d'accès de certaines requêtes. Par exemple, Oracle offre la possibilité de ranger physiquement plusieurs tables dans un "*cluster*" pour accélérer les jointures de ces tables. Quand des tables sont enregistrées dans un cluster, les valeurs de certaines colonnes sont partagées entre ces tables. Par exemple, pour la base étudiée dans ce cours, on peut réunir en un même cluster les tables EMP et DEPT, en partageant les valeurs des colonnes "dept" des deux tables. On trouvera alors réunis en des places contiguës sur le disque la ligne du département 10 de la table DEPT et les lignes de la table EMP concernant les employés du département 10.

6.6 Modules d'utilisation des SGBD

Les SGBD sont accompagnés de produits qui offrent de nombreux services à l'utilisateur, au développeur et à l'administrateur. En particulier,

- interface pour l'interrogation en direct de la base,
- générateur de formulaires pour la saisie et l'interrogation des données,
- générateur de menus,
- générateur d'états imprimés,
- générateur d'applications à partir d'analyses conceptuelles,
- langages dits de 4ème génération pour développer rapidement des applications qui utilisent une base de données,
- interface avec les langages de troisième génération (langage C, Fortran, Ada, etc.),
- interfaces avec les applications de bureautique (tableurs en particulier).

La figure 6.1 donne un aperçu des modules offerts par Oracle.

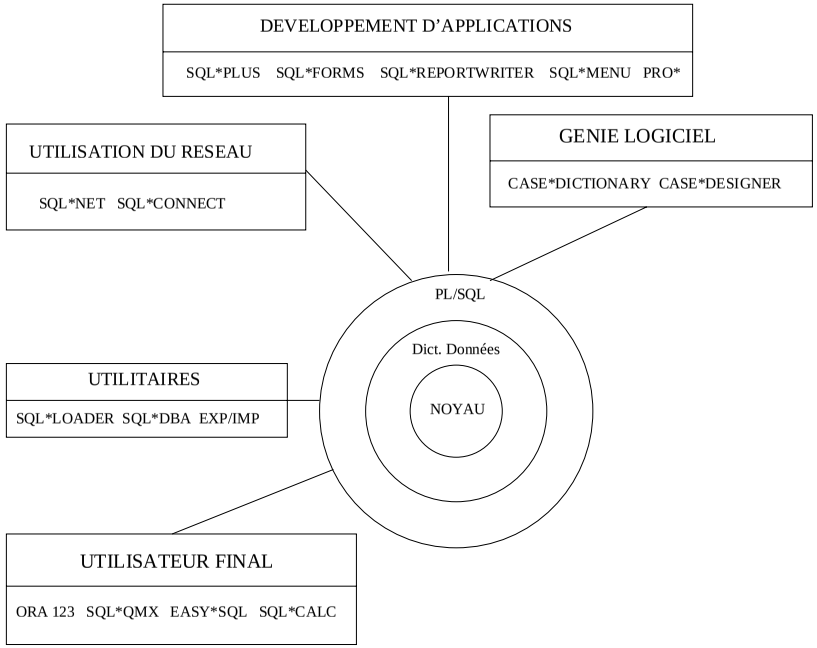

FIG. 6.1 – *Produits offerts par le SGBD Oracle*

Chapitre 7

Implantation d'un SGBD

Ce chapitre donne quelques notions sur les moyens mis en œuvre dans les SGBD pour satisfaire les besoins décrits dans le chapitre 6 (intégrité des données, sécurité, résistance aux pannes, etc.). Elle décrit plus particulièrement les solutions retenues par le SGBD Oracle (version 7).

La figure 7.1 représente un schéma de l'architecture du SGBD Oracle. Les différents éléments sont décrits dans les sections suivantes.

7.1 Fichiers de la base

Tous les grands SGBD gèrent eux-mêmes l'enregistrement des données sur les mémoires auxiliaires (disques le plus souvent). Les performances sont ainsi meilleures que s'ils utilisaient simplement le système d'exploitation et son système de fichiers. En général, ils se réservent de très gros fichiers du système d'exploitation et ils gèrent eux-mêmes cet espace pour allouer de la place aux tables, index et autres objets manipulés.

Un autre avantage est la plus grande facilité pour le portage sur des systèmes d'exploitation différents.

7.2 Processus clients et serveurs

L'utilisation d'un SGBD s'appuie sur le mode client-serveur : les applications des utilisateurs qui accèdent à la base sont clientes de processus serveurs lancés en arrière-plan par le SGBD.

Les applications clientes n'ont pas d'accès direct aux données de la base. Elles transmettent leurs requêtes (SQL pour Oracle et la plupart des SGBD relationnels) aux processus serveurs du SGBD. Un processus serveur peut prendre en charge l'interface entre la base et un ou plusieurs processus clients. Il y a en général plusieurs processus serveurs qui fonctionnent en même temps.

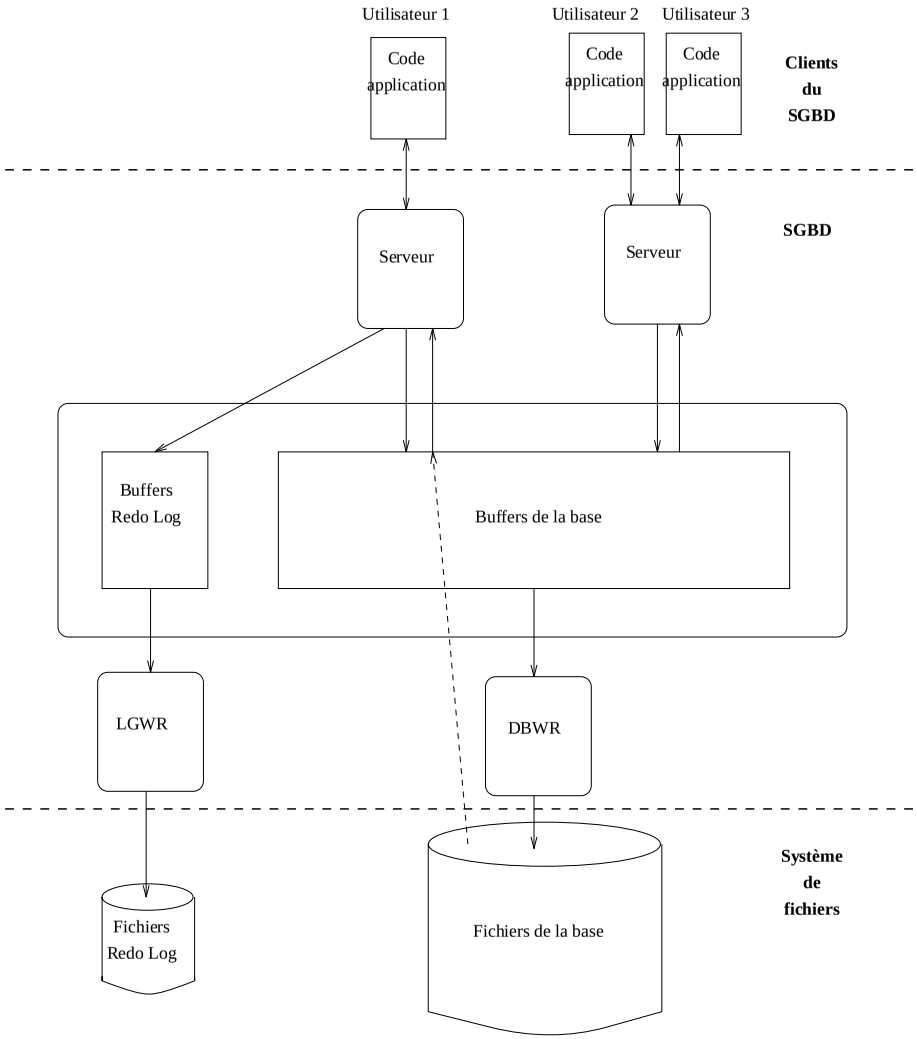

FIG. 7.1 – *Architecture système du SGBD Oracle*

Un processus serveur exécute les requêtes des clients dont il a la charge et en particulier

- transmet aux clients les résultats des requêtes de consultation de la base (les données seront lues en mémoire centrale dans les buffers de la base si elles y sont déjà),
- dépose dans les buffers de la base (en mémoire centrale) les données ajoutées ou modifiées par les requêtes de manipulation des données.

7.3 Écriture des données dans la base

Un seul processus ("*Data Base Writer*", DBWR) a la charge d'enregistrer dans les fichiers de la base les données déposées dans les buffers par les processus serveurs. Le fait qu'un seul processus soit chargé de cet enregistrement facilite le maintien de l'intégrité des données.

Pour des raisons de performance cette écriture se fait de manière asynchrone par rapport aux actions des serveurs. Il y a, par exemple, écriture quand les buffers sont pleins. L'écriture des différentes entrées des buffers se fait par l'algorithme LRU (*Least Recently Used*) et ne tient pas compte des COMMIT : les entrées qui sont enlevées des buffers et enregistrées dans la base sont celles qui ont été les moins récemment utilisées. Une entrée très utilisée peut donc être enregistrée (physiquement sur le disque) dans la base que longtemps après son écriture dans les buffers de la base.

7.4 Fichiers "log"

Parallèlement à l'enregistrement des données, les processus serveurs enregistrent toutes les actions effectuées sur les tables de la base (et dans les segments de rollback), dans des fichiers indépendants des fichiers de la base, appelés fichiers "log" ("Redo Log" dans Oracle).

Cette section décrit l'utilisation des fichiers "log" dans Oracle.

Comme pour les données de la base, les données enregistrées dans les fichiers "Redo Log" sont d'abord déposées en mémoire centrale dans des buffers et enregistrées dans les fichiers par un seul processus ("Log Writer", LGWR dans Oracle). Ces enregistrements ont lieu régulièrement, par exemple quand les buffers sont pleins, et au moins dès qu'il y a un COMMIT.

L'enregistrement d'un COMMIT dans les fichiers "Redo Log" est l'événement qui "fait foi" pour la confirmation d'une transaction. Chaque COMMIT déclenche une écriture immédiate sur le disque des entrées du buffer des fichiers "Redo Log" correpondant à la transaction validée. Cette écriture est rapide car séquentielle ; l'écriture des modifications dans la base même nuirait aux performances car le mécanisme d'écriture dans la base est complexe.

Remarque 7.1

Oracle écrit dans les fichiers "Redo Log" d'une façon circulaire : les fichiers "Redo Log" ont une certaine taille ; quand ils sont remplis, les nouvelles entrées écrasent les entrées les plus anciennes.

Une option d'Oracle permet de conserver sur disque ou sur bande les fichiers "Redo Log" quand ils sont pleins et avant que de nouvelles entrées n'y soient enregistrées. Cet archivage permet de remettre une base comme elle était au moment de la panne, à partir de la dernière sauvegarde (voir 6.3.3).

Que se passe-t-il s'il y a une panne alors que les fichiers "Redo Log" ont enregistrés un COMMIT, et avant que les données correspondant à la transaction de la base n'aient été écrites dans la base? Au redémarrage de la base, le SGBD s'en rend compte grâce à l'examen du moment où les écritures ont eu lieu. Il effectue donc l'enregistrement physique sur les disques des modification de la base. Si c'est un ROLLBACK qui a été enregistré et si des modifications ont déjà été enregistrées (par exemple, parce que les buffer des fichiers "Redo Log" étaient pleins), au redémarrage de la base, les données sont remises à leur état précédant la transaction, grâce aux segments de rollback.

7.5 Segments de Rollback

Les SGBD ont chacun leurs mécanismes pour effectuer un "rollback". Cette section décrit la méthode utilisée par la version 7 du SGBD Oracle.

Quand des données sont modifiées dans une base Oracle, toutes les informations qui permettraient de retrouver les anciennes données sont enregistrées dans la base dans ce qu'Oracle appelle des *segments de rollback*. Ces segments sont utilisés à deux occasions:

- pour annuler une transaction après une instruction ROLLBACK,
- tant qu'une transaction n'est pas confirmée par un COMMIT, les modifications qu'elle apporte aux données ne doivent pas apparaitre aux autres transactions. Les segments de Rollback sont utilisés par Oracle pour fournir à ces transactions les anciennes valeurs des données.

7.6 L'optimiseur de requêtes SQL

SQL est un langage non procédural. Par exemple, une instruction "SELECT" décrit les données recherchées sans indiquer la manière d'aller les rechercher dans la base.

L'optimiseur est le module du SGBD qui va concevoir un plan pour aller rechercher les données de la manière la plus efficace (trouver toutes les données, ou la première ligne, en le moins de temps possible). En particulier, il cherchera à tirer bénéfice des index.

L'optimiseur utilise des règles de simplification et d'optimisation implantées par les concepteurs du SGBD et les données statistiques sur les tables (par exemple, nombre de lignes, nombre de valeurs différentes dans les colonnes).

7.7 Bases de données réparties

Cette section introduit quelques notions et problèmes liés à la répartition des processus et des données sur plusieurs sites.

Chaque site gère localement les données qui sont enregistrées localement.

La transparence de l'implantation des données doit être recherchée: les utilisateurs n'ont pas à savoir où sont enregistrées les données quand ils écrivent leurs requêtes. Les SGBD actuels n'offrent pas une transparence complète mais les développeurs d'applications peuvent ajouter aisément (par exemple, en utilisant des synonymes ou des vues) une couche supplémentaire pour que l'utilisateur des applications n'ait pas à connaitre l'emplacement des données qu'il cherche.

Si toutes la base est réunie en un seul lieu, les accès distants à cette base ne posent pas de problèmes complexes. Les échanges entre les processus des applications et les serveurs du SGBD (voir figure 7.1) sont conçus sur le mode client/serveur et la seule différence avec les accès locaux est que la communication passe par un réseau en utilisant le protocole du réseau.

Les difficultés apparaissent lorsque les données sont réparties sur plusieurs sites.

Pour des raisons de performances, les données les plus souvent utilisées dans un site mais qui sont enregistrées sur un autre site peuvent être recopiées sur le site local, à intervalles réguliers. La gestion des recopies est à la charge des administrateurs des différents sites mais peut être automatisée, au moins partiellement.

Une difficulté de l'implantation des bases de données réparties est l'implantation des COMMIT et ROLLBACK. En effet, une panne du réseau ou sur un des sites au mauvais moment peut faire que le COMMIT d'une transaction qui met en jeu plusieurs sites ne peut être que partiellement réalisé, ce qui évidemment va à l'encontre de la notion de transaction. Pour éviter ce problème, les SGBD répartis utilisent le *COMMIT à 2 phases*. Un des sites (le plus souvent celui qui a reçu l'ordre de COMMIT) est le coordinateur de la manœuvre qui se déroule en deux étapes:

1. Le coordinateur ordonne à tous les SGBD locaux de "préparer la part" qui lui revient dans le COMMIT (confirmer les modifications qui ont eu lieu sur les données enregistrées localement). Quand il est prêt à exécuter le COMMIT, chaque site local envoie un message au coordinateur pour l'informer. "Être prêt" signifie être certain que sa part du COMMIT sera complètement exécutée si le coordinateur décide un COMMIT dans la deuxième phase, même si une panne se produisait. Dans le cas où le SGBD est Oracle, cela consiste essentiellement à écrire les actions nécessaires dans les fichiers "log". Ainsi, en cas de panne durant l'exécution du COMMIT, au moment de la reprise après panne (voir 7.4) le SGBD irait s'informer auprès du coordinateur de sa décision concernant cette transaction (confirmation ou annulation) et un COMMIT ou un ROLLBACK serait exécuté.
2. Si tous les sites impliqués dans la transaction informent le coordinateur qu'ils sont prêts, celui-ci inscrit le COMMIT dans son fichier "log" et il ordonne à tous les sites impliqués d'exécuter le COMMIT.

Si un des sites ne répond pas ou indique qu'il ne peut exécuter le COMMIT, le coordinateur envoie un ordre de ROLLBACK à tous les sites impliqués dans la transaction. Si une panne survient sur un des sites au moment d'exécuter le COMMIT, celui-ci sera exécuté lors de la reprise après panne : le SGBD local ira s'informer auprès du coordinateur de sa décision pour la transaction (COMMIT ou ROLLBACK) et il utilisera les informations qu'il aura enregistrées dans ses fichiers "log" lors de l'étape précédente pour exécuter le COMMIT.

En fonctionnement normal ce COMMIT à deux phases est transparent pour l'utilisateur qui, le plus souvent, ne sait même pas qu'il travaille sur des données réparties sur plusieurs sites.

Index