Stéphane Crozat

NF17 1

Cours général
1

Université de Technologie de Compiègne
Génie Informatique
http://www4.utc.fr/~nf17

COURS GÉNÉRAL

NF17

Conception de bases de données

Stéphane Crozat

Version 4.0 du 31 janvier 2006

Edité par : SCENARI

Entête pédagogique du module

Objectifs pédagogiques

Maîtriser les outils méthodologiques pour la conception de bases de données
Savoir mettre en pratique les outils technologiques classiques pour la réalisation de bases de données
Acquérir une expérience de conception et de réalisation informatique en équipe à travers un projet de développement d'un site web
Découvrir des projets industriels réels en rapport avec les bases de données
Développer un esprit critique face aux technologies (veille, évaluation, choix)
Faire réfléchir sur ce qu'est le métier d'ingénieur (articulation méthode-technique)

Résumé

L'objectif de l'UV est de d'amener les étudiants à maîtriser la conception de bases de données relationnelles et relationnelles-objet. Cette maîtrise reposera sur des compétences méthodologiques de modélisation conceptuelle et logique, ainsi que sur des compétences technologiques de mise en oeuvre au sein de SGBD typiques (tels que Access, MySQL, Oracle et PosgreSQL) et à travers les langages couramment utilisés en BD (tels que SQL, PHP ou Java). Les étudiants mèneront tout au long du semestre un projet qui servira de cadre d'application en situation des concepts préalablement étudiés et expérimentés. Des présentations de projets réels de conception de BD seront également proposés par des intervenants praticiens de la discipline.

Table des matières

Introduction

Les BD [Base de Données] sont nées vers la fin des années 1960 pour combler les limites des systèmes de fichiers. Les BD relationnelles, issues de la recherche de Codd, sont celles qui ont connu le plus grand essor depuis plus de 20 ans, et qui reste encore aujourd'hui les plus utilisées. Le langage SQL [Structured Query Language] est une couche technologique, idéalement indépendante des implémentations des SGBDR [Système de Gestion de Bases de Données Relationnelles], qui permet de créer et manipuler des BD relationnelles.

Les usages de BD se sont aujourd'hui généralisés pour entrer dans tous les secteurs de l'entreprise, depuis les "petites" BD utilisées par quelques personnes dans un service pour des besoins de gestion de données locales, jusqu'aux "grosses" BD qui gèrent centralement pour toute l'entreprise des données partagées par tous les acteurs de l'entreprise. Parallèlement l'accroissement de l'utilisation du numérique comme outil de manipulation de toutes données (bureautique, informatique applicative, etc.) et comme outil d'extension des moyens de communication (réseaux) d'une part et les évolutions technologiques (puissance des PC, Internet, etc.) d'autre part ont à la fois rendu indispensable et complexifié la problématique des BD.
Les conséquences de cette généralisation et de cette diversification des usages se retrouvent dans l'émergence de solutions conceptuelles et technologiques nouvelles et sans cesse renouvellées.
Ce cours se proposera de donner les bases conceptuelles et technologiques qui permettront à l'ingénieur de mobiliser des compétences générales pour la réalisation de projets informatiques mobilisant les BD. Il ne cherchera pas, par contre, à fournir des compétences techniques spécialisées (tel ou tel SGBD), ni à de cas d'exploitation privilégié (Internet, gros systèmes, BD applicatives, etc.). Le travail de l'élève-ingénieur consistera donc, à partir des savoirs généraux et des savoir-faire particuliers acquis, à chercher à prendre du recul sur le domaine pour être capable, lorsqu'il sera en situation professionnelle réelle, d'apprendre rapidement les compétences spécifiques qui seront alors nécessaires dans son contexte d'application.
Ce support est structuré en 10 cours, correspondant aux 10 cours magistraux de l'UV NF17 de l'UTC, en commençant par les bases théoriques indispensables à la conception des bases de données, en continuant par l'apprentissage du language SQL incontournable dans ce domaine, pour prolonger sur l'application technologique à travers deux SGBD très différents (Access et Oracle) et pour finir sur des problématiques plus avancées des BD (le relationnel-objet et l'usage dans le contexte Web). Un complément de cours est donné sur MySQL, utilisé en projet.

- *Mes remerciements les plus sincères vont à Dritan Nace*, enseignant-chercheur à l'Université de Technologie de Compiègne, qui a enseigné NF17 avant moi jusqu'en 2002, et dont le fonds documentaire en terme de cours et d'exercices a été largement récupéré pour la constitution de ce support ; ainsi qu'à Yacine Challal, Achene Beneyache et Hamida Seba qui ont apporté leur contribution à travers quelques exercices et auto-évaluations. Mes remerciements enfin à l'ensemble des acteurs très dynamiques sur le Web qui m'ont aidé sans le savoir dans ce travail (on se reportera à la bibliographie pour les connaître).

- Ce support ne saurait être considéré comme un travail achevé ou suffisant pour un auto-apprentissage. Il s'agit de notes organisées destinées à accompagner le cours, ainsi que les activités de travaux dirigés de NF17. Il n'est certainement pas exempt d'erreurs et j'invite les lecteurs, étudiants et enseignants, à me faire part de leurs remarques pour m'aider dans ce travail sans fin d'amélioration de ce support.
- Ce support est disponible à l'adresse www4.utc.fr/~nf17.

Préambule

Licence

▲ IMG. 1 : CREATIVE COMMONS LICENCE - SOME RIGHTS RESERVED

Ce support a été réalisé avec les technologies SCENARI (www.utc.fr/ics/site_scenari).

Contact
Stéphane Crozat : stephane.crozat@utc.fr / www.utc.fr/ics/~stc

Cours général

I

Modélisation

La *modélisation* est l'étape *fondatrice* du processsus de conception de BD. Elle consiste à abstraire le problème réel posé pour en faire une reformulation qui trouvera une solution dans le cadre technologique d'un SGBD [Système de Gestion de Bases de Données]. Après avoir rappelé succintement les fondements et objectifs des SGBD, ce chapitre proposera les outils méthodologiques nécessaires à la modélisation, à travers les formalismes E-A [Entité-Association] et UML [Unified Modeling Language].

Chapitre A. Introduction aux bases de données

Objectifs pédagogiques

Comprendre ce qu'est un SGBD.
Comprendre l'utilité des BD.
Connaître les différences entre modèle conceptuel, modèle logique et implémentation phyique.
Comprendre l'importance de la modélisation conceptuelle.

Section A1. Vue d'ensemble

1. Qu'est ce qu'une BD ?

Base de données
Une BD est un ensemble volumineux, structuré et minimalement redondant de données, reliées entre elles, stockées sur supports numériques centralisés ou distribués, servant pour les besoins d'une ou plusieurs applications, interrogeables et modifiables par un ou plusieurs utilisateurs travaillant potentiellement en parallèle.

Exemple : Compagnie aérienne
Une BD de gestion de l'activité d'une compagnie aérienne concernant les voyageurs, les vols, les avions, le personnel, les réservations, etc. Une telle BD pourrait permettre la gestion des réservations, des disponibilités des avions en fonction des vols à effectuer, des affectation des personnels volants, etc.

2. Qu'est ce qu'un SGBD ?

Système de Gestion de Bases de Données
Un SGBD est un logiciel qui prend en charge la structuration, le stockage, la mise à jour et la maintenance d'une base de données. Il est l'unique interface entre les informaticiens et les données (définition des schémas, programmation des applications), ainsi qu'entre les utilisateurs et les données (consultation et mise à jour).

Exemples de SGBD

- Oracle est un SGBD relationnel (et Relationnel-Objet dans ses dernières versions) très reconnu pour les applications professionnelles.
- MySQL est un SGBD relationnel libre (licence GPL et commerciale), simple d'accès et très utilisé pour la réalisation de sites Web dynamiques. Depuis la version 4 MySQL implémente la plupart des fonctions attendues d'un SGBD relationnel.
- PosgreSQL est un SGBD relationnel et relationnel-objet très puissant qui offre une alternative open-source aux solutions commerciales comme Oracle ou IBM.
- Access est un SGBD relationnel Microsoft, qui offre une interface conviviale permettant de concevoir rapidement des applications de petite envergure ou de réaliser des prototypes à moindre frais.

3. Pourquoi des SGBD ?

Jadis...

Avant l'avènement des SGBD, chaque application informatique dans l'entreprise impliquait sa propre équipe de développement, ses propres supports physiques, ses propres fichiers, ses propres normes, ses propres langages, etc.

Conséquences...

L'existence conjointe et croissante de ces applications indépendantes a des effets négatifs, tels que :

- La multiplication des tâches de saisie, de développement et de support informatique
- La redondance anarchique des informations dans les fichiers
- L'incohérence des versions simultanées de fichiers
- La non-portabilité des traitements en raison des différences dans les formats et langages.
- La multiplication des coûts de développement et de maintenance des applications.

Problèmes...

Les conséquences précédemment citées se répercutent sur l'entreprise en générant des problèmes humains et matériels.

Coûts en personnels qualifiés et en formations

- Remise des pouvoirs de décision entre les mains de spécialistes informatiques
- Tout changement matériel ou logiciel a un impact sur les applications
- Tout changement de la structure des données nécessite de modifier les programmes

Or...

En réalité les applications ne sont jamais totalement disjointes, des données similaires (le coeur de l'information d'entreprise) sont toujours à la base des traitements.

On peut citer typiquement :

- Les données comptables
- Les données clients et fournisseurs
- Les données relatives à la gestion des stocks
- Les données relatives aux livraisons
- Les données marketting et commerciales
- Les données relatives au personnel
- etc.

4. Caractéristiques des SGBD

La conception d'un système d'information pour être rationnelle à l'échelle d'une entreprise se doit d'adopter un certain nombre de principes, tels que :

- Une description des données indépendante des traitements
- Une maintenance de la cohérence de données
- Le recours à des langages non procéduraux, interactifs et structurants

Dans ce cadre les SGBD se fixent les objectifs suivants :

- **Indépendance physique des données**

  Le changement des modalités de stockage de l'information (optimisation, réorganisation, segmentation, etc.) n'implique pas de changements des programmes.

- **Indépendance logique des données**

  L'évolution de la structure d'une partie des données n'influe pas sur l'ensemble des données.

- **Manipulation des données par des non-informaticiens**

  L'utilisateur n'a pas à savoir comment l'information est stockée et calculée par la machine, mais juste à pouvoir la rechercher et la mettre à jour à travers des IHM [Interface Homme Machine] ou des langages assertionnels simples.

- **Administration facilitée des données**

  Le SGBD fournit un ensemble d'outils (dictionnaire de données, audit, tunning, statistiques, etc.) pour améliorer les performance et optimiser les stockages.

- **Optimisation de l'accès aux données**

  Les temps de réponse et de débits globaux sont optimisés en fonctions des questions posées à la BD.

- **Contrôle de cohérence (intégrité sémantique) des données**

  Le SGBD doit assurer à tout instant que les données respectent les règles d'intégrité qui leurs sont imposées.

- **Partageabilité des données**

  Les données sont simultanéments consultables et modifiables.

- **Sécurité des données**

  La confidentialité des données est assurée par des systèmes d'authentification, de droits d'accès, de cryptage des mots de passe, etc.

- **Sûreté des données**

  La persistence des données, même en cas de panne, est assurée, grâce typiquememnt à des sauvegardes et des journaux qui gardent une trace persistante des opérations effectuées.

Section A2. Notions générales

1. Notion de données

Type de données
Ensemble d'objets qui possèdent des caractéristiques similaires et manipulables par des opérations identiques.

- *Synonyme : Classe.*

Données
Elément effectif, réel, correspondant à une type de données.

- *Synonymes : Occurence, Instance.*

Exemple : Type de données

- Entier = { 0, 1, 2, ... , N }
- Véhicule = (immatriculation, marque, type, couleur)

Exemple : Données

- L'entier 486
- Le véhicule (460HP59, Renault, Megane, Jaune)

2. Notion de modèle de données

Modèle de données

Ensemble de concepts et de règles de composition de ces concepts permettant de décrire des données (Gardarin, 1999).
Un modèle est souvent représenté au moyen d'un formalisme graphique permettant de décrire les données (ou plus précisément les types de données) et les relations entre les données.

On distingue trois niveaux de modélisation pour les bases de données :

- **Le modèle conceptuel**

  Il permet de décrire le réel selon une approche ontologique, sans prendre en compte les contraintes techniques.

- **Le modèle logique**

  Il permet de décrire une solution, en prenant une orientation informatique générale (type de SGBD typiquement), mais indépendamment de choix d'implémentation précis.

- **Le modèle physique**

  Il correspond aux choix techniques, en terme de SGBD choisi et de sa mise en oeuvre (programmation, optimisation, etc.).

Exemple de formalisme de modélisation conceptuelle

- Le modèle Entité-Association (Chen, 1976) a été le plus répendu dans le cadre de la conception de bases de données.
- Le modèle UML, qui se généralise pour la conception en informatique, se fonde sur une approche objet.

Exemple de formalisme de modélisation logique

- Le modèle relationnel est le modèle dominant.
- Le modèle relationnel-objet (adaptation des modèles relationnel et objet au cadre des SGBD) est actuellement en pleine croissance.
- Le modèle objet "pur" reste majoritairement au stade expérimental et de la recherche.
- Des modèles plus anciens (hiérarchique, réseau, etc.) ne sont plus guère utilisés aujourd'hui.

3. Notion de schéma de données

Schéma de données

Description, au moyen d'un langage formel, d'un ensemble de données dans le contexte d'une BD.

Un schéma permet de décrire la structure d'une base de données, en décrivant l'ensemble des types de données de la base. L'occurence d'une base de données est constituée de l'ensemble des données correspondant aux types du schéma de la base.

Exemple : Schéma de base de données

```
Etudiant (NumEtud, nom, ville)
Module(NumMod, titre)
Inscription(NumEtud, NumMod, date)
```

Exemple : Instance de base de données

```
Etudiant (172, 'Dupont', 'Lille')
Etudiant (173, 'Durand', 'Paris')
Etudiant (174, 'Martin', 'Orléans')
Module(1, 'SGBD')
Module(1, 'Systèmes d'exploitation')
Inscription(172, 1, 2002)
Inscription(172, 2, 2002)
Inscription(173, 1, 2001)
Inscription(174, 2, 2002)
```

On distingue trois niveaux d'abstraction de schémas :

- **Le niveau conceptuel**

  Il permet de décrire les entités et les associations du monde réel. Il s'agit du schéma global de la base de données, il en propose une vue canonique.

  Le niveau conceptuel correspond au modèle conceptuel.

- **Le niveau externe**

  Il permet de décrire les entités et les associations du monde réel, mais vues d'un utilisateur ou d'un groupe d'utilisateurs particuliers (on parle d'ailleurs également de "vue" pour un shéma externe). Il s'agit d'une restriction du schéma conceptuel orientée vers un usage précis. Il existe généralement plusieurs schémas externes pour un même schéma conceptuel.

  Le niveau externe correspond à un sous ensemble du modèle conceptuel restreint aux points de vue de certains utilisateurs.

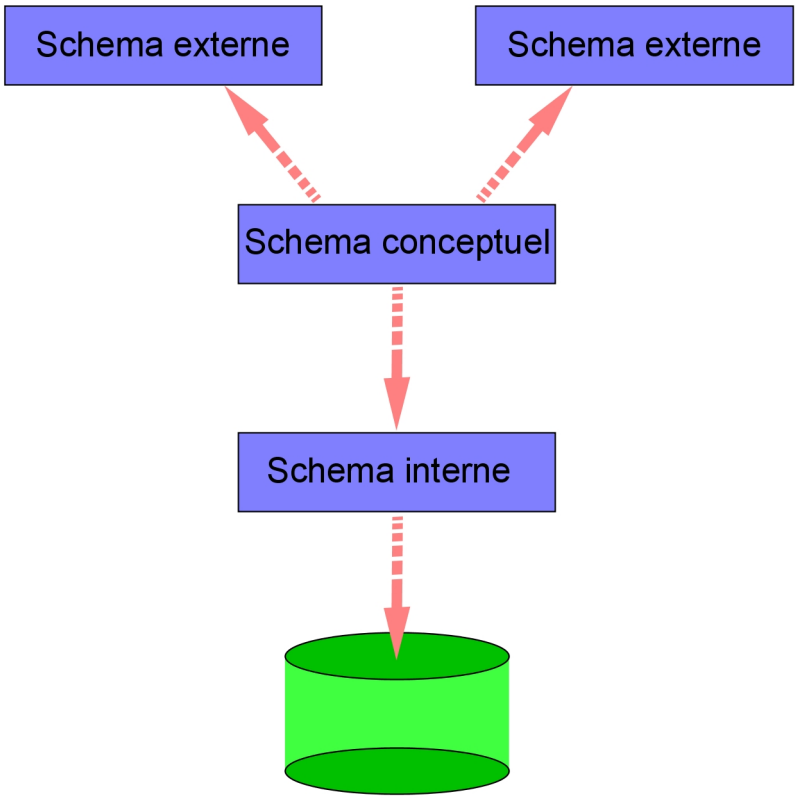
- **Le niveau interne**

  Il correspond à l'implémentation physique des entités et associations dans les fichiers de la base.

  Le niveau interne correspond aux modèles logiques et physiques.

Remarque : ANSI/X3/SPARC

Les trois niveaux, conceptuel, externe et interne, sont les trois niveaux distingués par le groupe de normalisation ANSI/X3/SPARC en 1975.



▲ SCH. 1 : LES TROIS NIVEAUX DE SCHÉMA SELON ANSI/X3/SPARC

4. Notion de langage de données

Langage de données
Langage informatique permettant de décrire et de manipuler les schémas d'une BD d'une une manière assimilable par la machine.

- *Synonyme : Langage orienté données.*

Exemple : SQL
SQL est le langage orienté données consacré aux SGBD relationnels et relationnels-objet.

Un langage de données peut être décomposé en trois sous langages :

- **Le Langage de Définition de Données**

  Le LDD [Langage de Définition de Données] permet d'implémenter le schéma conceptuel (notion de table en SQL) et les schémas externes (notion de vue en SQL).

- **Le Langage de Contrôle de Données**

  Le LCD [Langage de Contrôle de Données] permet d'implémenter les droits que les utilisateurs ont sur les données et participe donc à la définition des schémas externes.

- **Le Langage de Manipulation de Données**

  Le LMD [Langage de Manipulation de Données] permet l'interrogation et la mise à jour des données. C'est la partie du langage indispensable pour exploiter la BD et réaliser les applications.

Exemple : Définition de données en SQL

```
CREATE TABLE Etudiant (
   NumEtu : integer,
   Nom : string,
   Ville : string)
```

Cette instruction permet de créer une relation "Etudiant" comportant les propriétés "NumEtu", "Nom" et "Ville".

Exemple : Contrôle de données en SQL

```
GRANT ALL PRIVILEGES ON Etudiant FOR 'Utilisateur'
```

Cette instruction permet de donner tous les droits à l'utilisateur "Utilisateur" sur la relation "Etudiant".

Exemple : Manipulation de données en SQL

```
SELECT Nom
FROM Etudiant
WHERE Ville = 'Compiègne'
```

Cette instruction permet de rechercher les noms de tous les étudiants habitant la ville de Compiègne.

5. Notion d'administration de données

Administrateur

Personne ou groupe de personnes responsables de la définition des différents niveaux de schéma.

On distingue un type d'administrateur par niveau de schéma :

- L'administrateur entreprise est en charge de la gestion du schéma conceptuel et des règles de contrôle des données.
- L'administrateur de données est en charge de la gestion des schémas externes et de leur correspondance avec le schéma conceptuel.
- L'administrateur base de données est en charge de la gestion du schéma interne et de sa correspondance avec le schéma conceptuel.

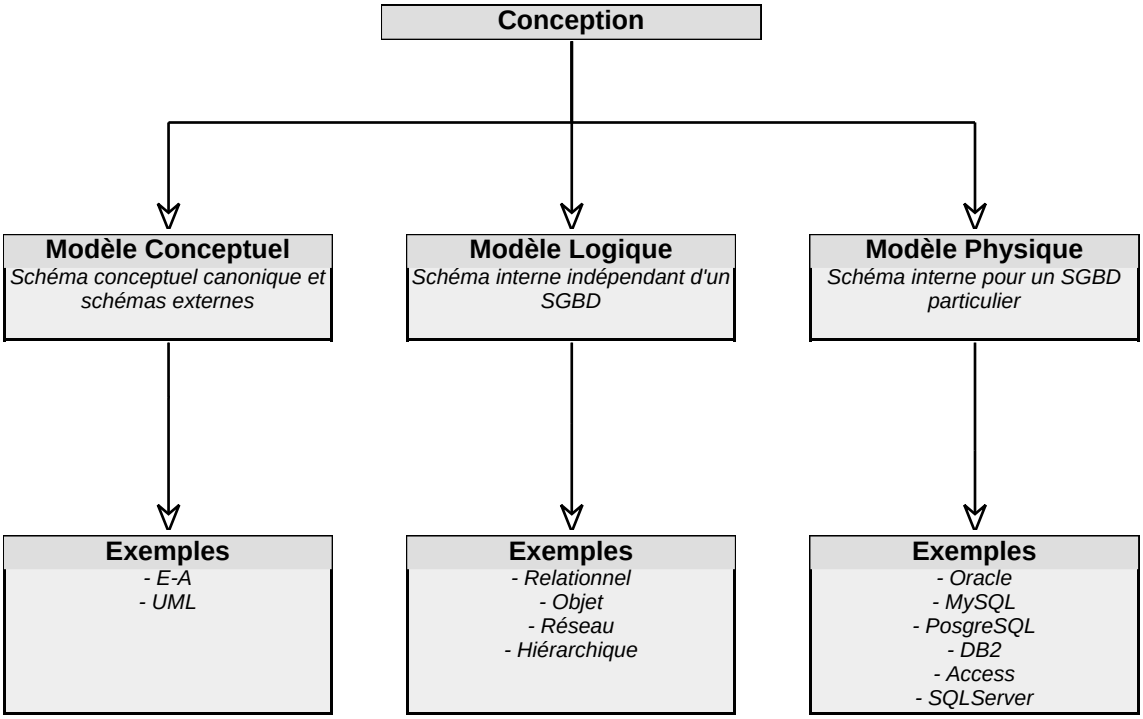
Dictionnaire des données

Le dictionnaire de données d'un SGBD contient les informations relatives aux schémas et aux droits de toutes les bases de données existantes au sein de ce SGBD. Il s'agit d'un outil fondamental pour les administrateurs.

Les dictionnaires de données sont généralement implémentés sous la forme d'une base de données particulière du SGBD, ce qui permet de gérer les données relatives aux bases de données de la même façon que les autres données de l'entreprise (i.e. dans une base de données).

- *Synonymes : Catalogue des données, Métabase.*

En résumé...



* *
*

Les SGBD assurent la gestion efficace et structurée des données partagées. Leur conception repose sur une approche à trois niveaux : conceptuel et externe, logique, physique.

Chapitre B. Le niveau conceptuel

Objectifs pédagogiques

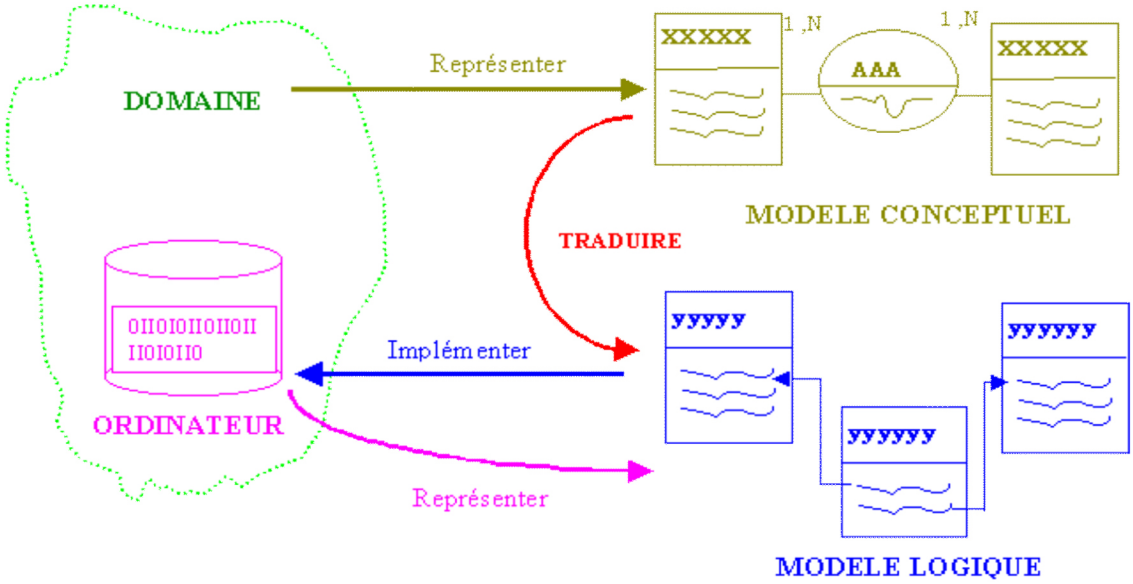
Comprendre la méthodologie de conception d'une BD.
Savoir réaliser et interpréter un modèle conceptuel.
Maîtriser la modélisation E-A.
Maîtriser la modélisation UML dans le cas particulier de la conception de BD.

Section B1. Les méthodes de conception de bases de données

1. Méthodologie de conception d'une base de données

Etapes de la conception d'une base de données

1. Analyse de la situation existante et des besoins
2. Création d'une série de modèles conceptuels (canonique et vues externes) qui permettent de représenter tous les aspects importants du problème
3. Traduction des modèles conceptuels en modèle logique et optimisation (normalisation) de ce modèle logique
4. Implémentation d'une base de données dans un SGBD, à partir du modèle logique



▲ IMG. 2 : PROCESSUS DE CONCEPTION D'UNE BASE DE DONNÉES

L'importance de l'étape d'analyse

La première étape de la conception repose sur l'analyse de l'existant et des besoins. De la qualité de la réalisation de cette première étape dépendra ensuite la pertinence de la base de données par rapports aux usages. Cette première étape est donc essentielle et doit être menée avec soins.
Si la première étape est fondamentale dans le processus de conception, elle est aussi la plus délicate. En effet, tandis que des formalismes puissants existent pour la modalisation conceptuel puis pour la modélisation logique, la perception de l'existant et des besoins reste une étape qui repose essentiellement sur l'expertise d'analyse de l'ingénieur.

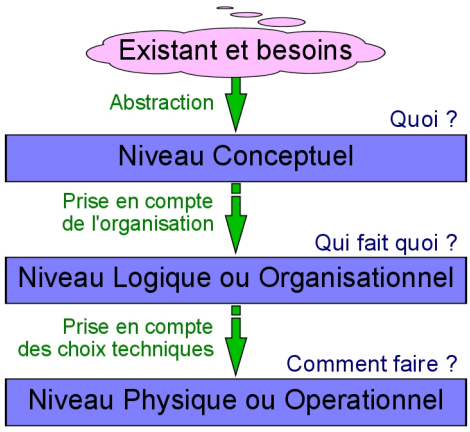
L'importance de l'étape de modélisation conceptuelle

Etant donnée une analyse des besoins correctement réalisée, la seconde étape consiste à la traduire selon un modèle conceptuel. Le modèle conceptuel étant formel, il va permettre de passer d'une spécification en langage naturel, et donc soumise à interprétation, à une spécification non ambigüe. Le recours aux formalismes de modélisation tels que E-A ou UML est donc une aide fondamentale pour parvenir à une représentation qui ne sera plus liée à l'interprétation du lecteur.
La traduction d'un cahier des charges spécifiant l'existant et les besoins en modèle conceptuel reste néanmoins une étape délicate, qui va conditionner ensuite l'ensemble de l'implémentation informatique. En effet les étape suivantes sont plus mécaniques, dans la mesure où un modèle logique est déduit de façon systématique du modèle conceptuel et que l'implémentation logicielle est également réalisée par traduction directe du modèle logique.

Remarque : Les étapes de traduction logique et d'implémentation
Des logiciels spécialisés (http://www.sybase.com/products/enterprisemodeling pour la modélisation E-A ou www.objecteering.com pour la modélisation UML) sont capables à partir d'un modèle conceptuel d'appliquer des algorithmes de traduction qui permettent d'obtenir directement le modèle logique, puis les instructions pour la création de la base de données dans un langage orienté données tel que SQL. L'existence de tels algorithmes de traduction montre que les étapes de traduction logique et d'implémentation sont moins complexes que les précédentes, car plus systématiques.
Néanmoins ces étapes exigent tout de même des compétences techniques pour optimiser les modéles logiques (normalisation), puis les implémentations en fonction d'un contexte de mise en oeuvre matériel, logiciel et humain.

2. La méthode MERISE et le modèle E-A

MERISE est une méthode d'analyse informatique particulièrement adaptée à la conception de bases de données.



▲ SCH. 2 : L'ANALYSE SELON MERISE

La méthode MERISE a pour fondement le modèle E-A, qui a fait son succès.

Les principales caractéristiques du modèle E-A sont :
- Une représentation graphique simple et naturelle
- Une puissance d'expression élevée pour un nombre de symboles raisonnables
- Une lecture accessible à tous et donc un bon outil de dialogue entre les acteurs techniques et non techniques
- Une formalisation non ambigüe et donc un bon outil de spécification détaillée

Remarque : L'arrivée d'UML
UML est un autre langage de modélisation, plus récent et couvrant un spectre plus large que les bases de données. En tant que standard de l'OMG [Object Management Group] et en tant qu'outil très utilisé pour la programmation orientée objet, il est très certainement amené à supplenter rapidement la modélisation E-A.

3. Eléments pour l'analyse de l'existant et des besoins

La phase d'analyse de l'existant et des besoins est une phase essentielle et complexe. Elle doit aboutir à des spécifications générales qui décrivent en langage naturel les données manipulées, et les traitements à effectuer sur ces données.

On se propose de donner une liste *non exhaustive* d'actions à mener pour rédiger de telles spécifications.

L'analyse de documents existants

La conception d'une base de données s'inscrit généralement au sein d'usages existants. Ces usages sont généralement, au moins en partie, instrumentés à travers des documents électroniques ou non (papier typiquement). Il est fondamental d'analyser ces documents et de recenser les données qu'ils manipulent.

Exemples de document existants

- Fichiers papiers de stockage des données (personnel, produits, etc.)
- Formulaires papiers d'enregistrement des données (fiche d'identification d'un salarié, fiche de description d'un produit, bon de commande, etc.)
- Documents électroniques de type traitement de texte (lettres, mailing, procédures, etc.)
- Documents électroniques de type tableurs (bilans, statistiques, calculs, etc.)
- Bases de données existantes, à remplacer ou avec lesquelles s'accorder (gestion des salaires, de la production, etc.)
- Intranet d'entreprise (information, téléchargement de documents, etc.)
- etc.

Le recueil d'expertise métier

Les données que la base va devoir manipuler sont toujours relatives aux métiers de l'entreprise, et il existe des experts qui pratiquent ces métiers. Le dialogue avec ces experts est une source importante d'informations. Il permet également de fixer la terminologie du domaine.

Exemples d'experts à consulter

- Praticiens (secrétaires, ouvrier, contrôleurs, etc.)
- Cadres (responsables de service, contremaîtres, etc.)
- Experts externes (clients, fournisseurs, etc.)
- etc.

Le dialogue avec les usagers

La base de données concerne des utilisateurs cibles, c'est à dire ceux qui produiront et consommeront effectivement les données de la base. Il est nécessaire de dialoguer avec ces utilisateurs, qui sont les détenteurs des connaissances relatives aux besoins réels, liés à leur réalité actuelle (aspects de l'organisation fonctionnant correctement ou défaillants) et à la réalité souhaitée (évolutions, lacunes, etc.).

Exemples d'utilisateurs

- Personnes qui vont effectuer les saisies d'information (à partir de quelles sources ? Quelle est leur reponsabilité ? etc.)
- Personnes qui vont consulter les informations saisies (pour quel usage ? pour quel destinataire ? etc.)
- Personnes qui vont mettre à jour les informations (pour quelles raisons ? comment le processus est enclenché ? etc.)
- etc.

L'étude des autres systèmes informatiques existants

la base de données va généralement (et en fait quasi systématiquement aujourd'hui) s'insérer parmi un ensemble d'autres logiciels informatiques travaillants sur les données de l'entreprise. Il est important d'analyser ces systèmes, afin de mieux comprendre les mécanismes existants, leurs forces et leurs lacunes, et de préparer l'intégration de la base avec ces autres systèmes. Une partie de ces systèmes seront d'ailleurs souvent également des utilisateurs de la base de données, tandis que la base de données sera elle même utilisatrice d'autre systèmes.

Exemples d'autres systèmes coexistants

- Autres bases de données (les données sont elle disjointes ou partiellement communes avec celles de la base à concevoir ? quelles sont les technologies logicielles sur lesquelles reposent ces BD ? etc.)
- Systèmes de fichiers classiques (certains fichiers ont-ils vocations à être supplantés par la base ? à être générés par la base ? à alimenter la base ? etc.)
- Applications (ces applications ont elles besoins de données de la base ? peuvent-elles lui en fournir ? etc.)
- etc.

4. Le MCD

MCD

Le MCD [Modèle Conceptuel de Données] est l'élément le plus connu de MERISE et certainement le plus utile. Il permet d'établir une représentation claire des données du SI [Système d'Information] et définit les dépendances des données entre elles.

Exemple

Le modèle E-A est un formalisme de MCD.

Remarque

Un MCD est indépendant de l'état de l'art technologique. A ce titre il peut donc être mis en oeuvre dans n'importe quel environnement logiciel et matériel, et il devra être traduit pour mener à une implémentation effective.

5. Le MLD

On ne sait pas implémenter directement un modèle conceptuel de données dans une machine et il existe différentes sortes de SGBD qui ont chacun leur propre modèle : SGF [Système de Gestion de Fichiers] (qui ne sont pas vraiment des SGBD), SGBD hiérarchiques (organisés selon une arborescence), SGBD réseau (encore appelés CODASYL), SGBDR, SGBDOO [Système de Gestion de Bases de Données Orientées Objets], SGBDRO [Système de Gestion de Bases de Données Relationnelles-Objets], etc.

MLD

Un MLD [Modèle Logique de Données] est une *représentation* du système tel qu'il sera implémenté dans un ordinateur.

Exemple

Le modèle relationnel est un formalisme de MLD.

Remarque

Il ne faut pas confondre le MLD (relationnel par exemple) avec le MCD (E-A par exemple).

Il ne faut pas confondre le MLD avec son implémentation logicielle en machine (avec Oracle par exemple)

Section B2. Le modèle E-A

Le modèle E-A en bref

Le modèle E-A (ou E-R [Entity-Relationship] en anglais) permet la modélisation conceptuelle de données.

Il correspond au niveau conceptuel de la méthode MERISE (méthode d'analyse informatique), le MCD (Tardieu & al., 1983), (Tardieu & al., 1985).
La conception E-A est issue des travaux de Chen, (Chen, 1976) et se fonde sur deux concepts principaux et un troisième sous-jacent : l'entité, l'association et l'attribut ou propriété.

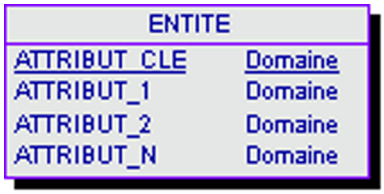
1. Entité

Entité

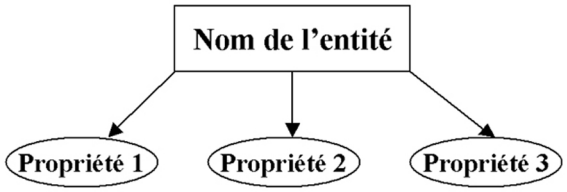
Une entité est un objet du monde réel avec une existence indépendante.

Une entité (ou type d'entité) est une chose (concrète ou abstraite) qui existe et est distinguable des autres entités.
L'occurrence d'une entité est un élément particulier correspondant à l'entité et associé à un élément du réel. Chaque entité a des propriétés (ou attributs) qui la décrivent. Chaque attribut est associé à un domaine de valeur. Une occurence a des valeurs pour chacun de ses attributs, dans le domaine correspondant.

Syntaxe



▲ IMG. 3 : NOTATION MERISE DE L'ENTITÉ



▲ IMG. 4 : NOTATION CHEN DE L'ENTITÉ

Remarque

- Un attribut est atomique, c'est à dire qu'il ne peut prendre qu'une seule valeur pour une occurence.
- Un attribut est élémentaire, c'est à dire qu'il ne peut être exprimé par (ou dérivé) d'autres attributs.
- Un attribut qui identifie de façon unique une occurence est appelé attribut *clé*.

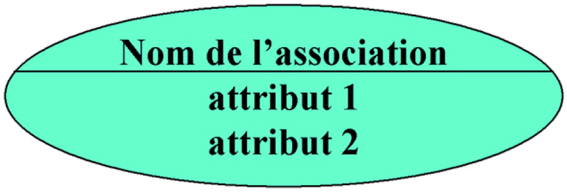
2. Association

Association

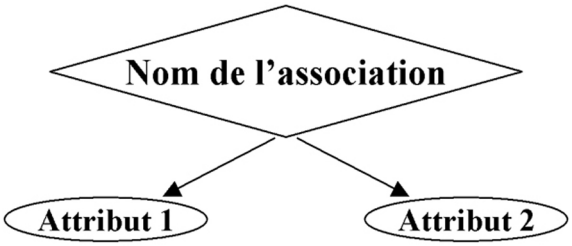
Une association (ou type d'association) représente un lien quelconque entre différentes entités.

Une occurrence d'une association est un élément particulier de l'association constitué d'une et une seule occurrence des objets participants à l'association. On peut définir des attributs sur les associations. Le degré d'une association est le nombre d'entités y participant (on parlera notamment d'association binaire lorsque deux entités sont concernées).

Syntaxe



▲ IMG. 5 : NOTATION MERISE DE L'ASSOCIATION



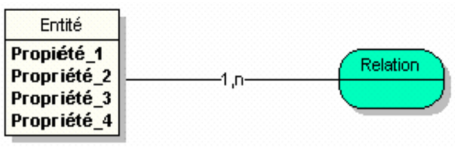
▲ IMG. 6 : NOTATION CHEN DE L'ASSOCIATION

Remarque
On peut avoir plusieurs associations différentes définies sur les mêmes entités.

3. Cardinalité d'une association

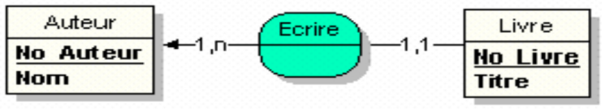
Cardinalité d'une association binaire
Pour les associations binaires la cardinalité minimale (resp. maximale) d'une association est le nombre minimum (resp. maximum) d'occurrences de l'entité d'arrivée associées à une occurrence de l'entité de départ.

Syntaxe



▲ IMG. 7 : NOTATION DE LA CARDINALITÉ

Exemple



▲ IMG. 8 : LIVRE-AUTEUR

Le diagramme E-A précédent exprime qu'un auteur peut avoir écrit plusieurs livres (mais au moins un), et que tout livre ne peut avoir été écrit que par un et un seul auteur.

Remarque : Trois grands types de cardinalité
Il existe trois grands types de cardinalité :

- Les associations 1:N (qui incluent les association 0,1:N)
- Les associations N:M
- Les associations 1:1 (qui incluent les association 0,1:1)

Les autres associations peuvent toujours être ramenées à des associations N:M (dans le cas général) ou à plusieurs associations 1:N (cas des associations 2:N ou 3:N par exemple).

4. Modèle E-A étendu

On peut étendre le modèle E-A "classique" de façon à accroitre son pouvoir de représentation. Cette extension du modèle E-A permet de favoriser la dimension conceptuelle et de s'approcher des représentations objet, telles que UML.

Attributs composites
Un attribut peut être composé hiérarchiquement de plusieurs autres attributs.

Exemple
Un attribut Adresse est composé des attributs Numéro, Rue, No_Appartement, Ville, Code_Postal, Pays.

Remarque
Le domaine d'un attribut composite n'est donc plus un domaine simple (entier, caractères, etc.).

Attributs multivalués
Tout attribut peut être monovalué ou multivalué.

Exemple
Les âges des enfants d’un employé.

Remarque
Un attribut multivalué n'est donc plus atomique.

Attributs dérivé
La valeur d'un attribut peut être dérivée d'une ou plusieurs autres valeurs d'attributs.

Exemple
L'âge d'une personne peut être dérivé de la date du jour et de celle de sa naissance.

Remarque
Un attribut dérivé n'est donc plus élémentaire.

Sous-type d'entité
Une entité peut-être définie comme sous-type d'une entité plus générale.

Exemple
Les entités Cadre et Technicien sont des sous-types de l'entité Employé.

Remarque
La notion de sous-type est équivalente à la notion d'héritage en modélisation objet.

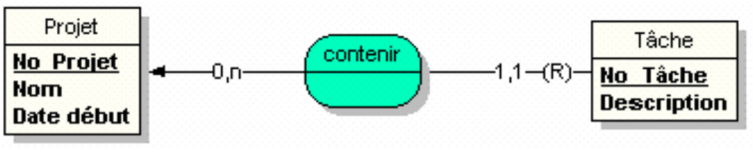
5. Entité de type faible

Certaines entités dites "faibles" n'existent qu'en référence à d'autres entités dites "identifiantes". L'entité identifiante est appelé "identifiant étranger" et l'association qui les unit "association identifiante".

Entité de type faible

Une entité de type faible, également appelée entité non identifiée, possède une clé locale (appelée identifiant relatif) qui permet d'identifier une de ses occurrences parmi l'ensemble des occurrences associées à une occurrence de l'entité identifiante. La clé complète d'une entité faible est la concaténation de la clé de l'entité identifiante et de sa clé locale.

Exemple



▲ IMG. 9 : ASSOCIATION IDENTIFIANTE

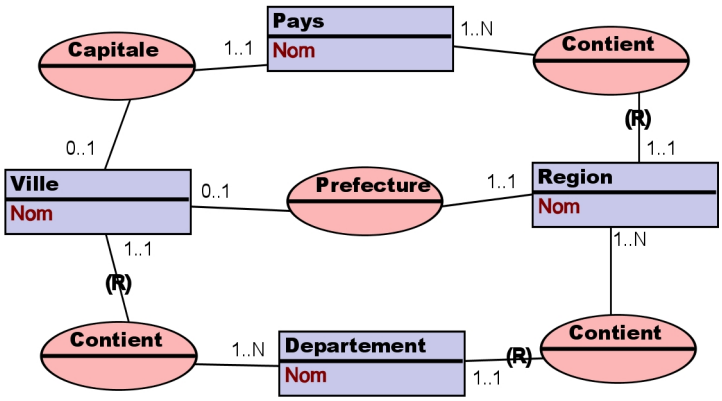
L'entité Tâche est complètement dépendante de l'entité Projet et sa clé locale (No_tâche) n'est pas suffisante à l'identifier de façon absolue.

Attention

Le repérage des entités de type faible est très important dans le processus de modélisation, il permet de réfléchir à la meilleure façon d'identifier de façon unique les entités et donc de trouver les meilleures clés. Notons de plus que le repérage d'entités faibles aura une influence importante sur le modèle relationnel résultant.

6. Illustration d'entités faibles

Exemple



▲ SCH. 3 : GÉOGRAPHIE

Explication

Dans le schéma ci-avant, on remarque que l'entité "ville" est faible par rapport à l'entité "département", qui est faible par rapport à "région", qui est faible par rapport à "pays". Cela signifie que la clé de ville, son nom, est une clé locale et donc que l'on considère qu'il ne peut pas y avoir deux villes différentes avec le même nom, *dans un même département*. Il est par contre possible de rencontrer deux villes différentes avec le même nom, dans deux départements différents. De la même façon chaque département possède un nom qui l'identifie de façon unique dans une région, et chaque région possède un nom qui l'identifie de façon unique dans un pays.
Si les entités n'étaient pas faibles, l'unicité d'un nom de ville serait valable pour l'ensemble du modèle, et donc, concrêtement, cela signifierai qu'il ne peut exister deux villes avec le même nom au monde (ni deux départements, ni deux régions).

Remarque
Notons pour terminer que, puisque "pays" n'est pas une entité faible, sa clé "nom" est bien unique pour l'ensemble du modèle, et donc cela signifie qu'il ne peut exister deux pays avec le même nom au monde.

Section B3. Les diagrammes de classes UML

Si le modèle dominant en conception de bases de données a longtemps été le modèle E-A, le modèle UML se généralise de plus en plus. Nous ne donnons ici (parmi l'ensemble des outils d'UML) qu'un aperçu du diagramme de classes, qui plus est limité aux aspects particulièrement utilisés en modélisation de bases de données.

Bref aperçu

UML est un langage de représentation destiné en particulier à la modélisation objet. UML est devenu une norme OMG en 1997[OMG est un organisme à but non lucratif créé en 1989 à l'initiative de sociétés comme HP, Sun, Philips, etc.].

UML propose un formalisme qui impose de "penser objet" et permet de rester indépendant d'un langage de programmation donné. Pour ce faire, UML normalise les concepts de l'objet (énumération et définition exhaustive des concepts) ainsi que leur notation graphique. Il peut donc être utilisé comme un moyen de communication entre les étapes de spécification conceptuelle et les étapes de spécifications techniques.
Dans le domaine des bases de données, UML peut être utilisé à la place du modèle E-A pour modéliser le domaine. De la même façon, un schéma conceptuel UML peut alors être traduit en schéma logique (relationnel ou relationnel-objet typiquement).

1. Classes

Classe
Une classe est un type abstrait caractérisé par des propriétés (attributs et méthodes) communes à un ensemble d'objets et permettant de créer des instances de ces objets, ayant ces propriétés.

Syntaxe

Nom de la classe
Attributs
Méthodes()

▲ IMG. 10 : REPRÉSENTATION UML D'UNE CLASSE

Exemple : La classe Voiture

Voiture
Marque : string Type : string NbPortes : integer Puissance : integer Kilométrage : integer
Rouler()

▲ IMG. 11 : EXEMPLE DE CLASSE REPRÉSENTÉE EN UML

Exemple : Une instance de la classe Voiture

L'objet V1 est une instance de la classe Voiture.

V1 : Voiture

- Marque : 'Citroën'
- Type : 'ZX'
- Portes : 5
- Puissance : 6
- Kilométrage : 300000

Remarque : Clé

Le repérage des clés n'est pas systématique en UML (la définition des clés se fera alors au niveau logique). On conseillera néanmoins de les représenter (en les soulignant dans le dessin). On évitera par contre d'ajouter des clés artificielles lorsqu'aucune clé n'est évidente.

Remarque

La modélisation sous forme de diagramme de classes est une modélisation statique, qui met en exergue la structure d'un modèle, mais ne rend pas compte de son évolution temporelle. UML propose d'autres types de diagrammes pour traiter, notamment, de ces aspects.

2. Attributs

Attribut

Un attribut est une information élémentaire qui caractérise une classe et dont la valeur dépend de l'objet instancié.

Remarque

- **Un attribut est typé**

  Le domaine des valeurs que peut prendre l'attribut est fixé a priori

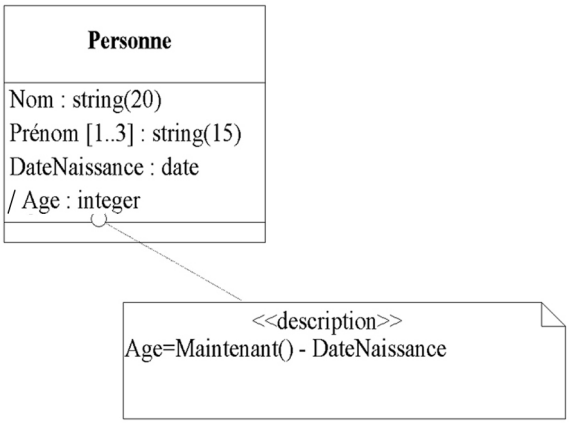
- **Un attribut peut être multivalué**

  Il peut prendre plusieurs valeurs distinctes dans son domaine

- **Un attribut peut être dérivé**

  Sa valeur alors est une fonction sur d'autres attributs de la classe (il peut donc aussi être représenté comme une méthode, et c'est en général préférable)

Exemple : La classe Personne



▲ IMG. 12 : REPRÉSENTATION D'ATTRIBUTS EN UML

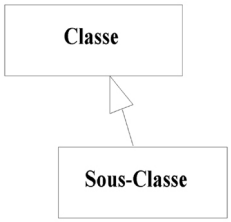
Dans cet exemple, les attributs Nom, Prénom sont de type string, l'un de 20 caractères et l'autre de 10,

tandis que DateNaissance est de type date et Age de type integer. Prénom est un attribut multivalué, ici une personne peut avoir de 1 à 3 prénoms. Age est un attribut dérivé, il peut être calculé par une fonction sur DateNaissance.

3. Héritage

Héritage
L'héritage est relation entre deux classes permettant d'exprimer que l'une est plus générale que l'autre. L'héritage implique une transmission automatique des propriétés (attributs et méthodes) d'une classe A à une classe A'.
Dire que A' hérite de A équivaut à dire que A' est une sous-classe de A. On peut également dire que A est une généralisation de A' et que A' est une spécialisation de A.

Syntaxe

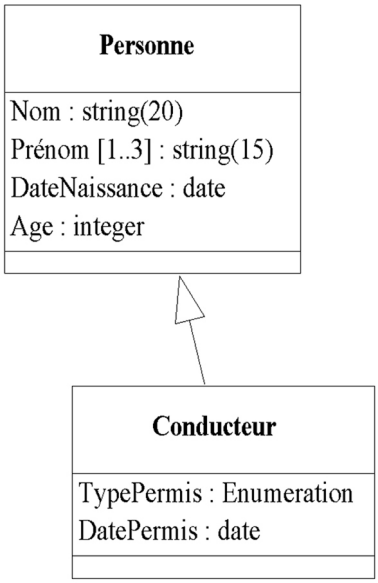


▲ IMG. 13 : NOTATION DE L'HÉRITAGE EN UML

Remarque
L'héritage permet de représenter la relation "est-un" entre deux objets.

Remarque
Outre qu'il permet de représenter une relation courante dans le monde réel, l'héritage a un avantage pratique, celui de factoriser la définition de propriétés identiques pour des classes proches.

Exemple : La classe Conducteur



▲ IMG. 14 : REPRÉSENTATION D'HÉRITAGE EN UML

Dans cet exemple la classe Conducteur hérite de la classe Personne, ce qui signifie qu'un objet de la classe conducteur aura les attributs de la classe Conducteur (TypePermis et DatePermis) mais aussi ceux

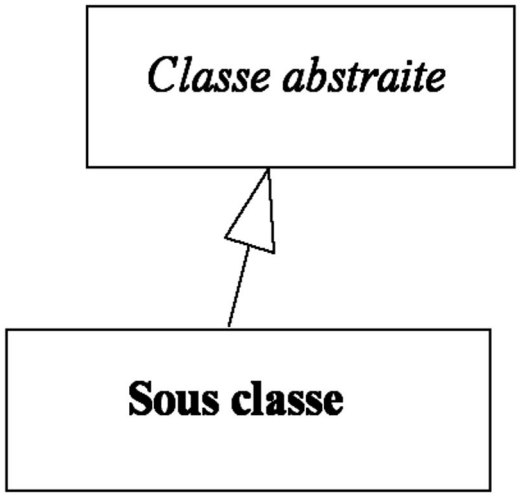
de la classe Personne (Nom, Prénom, DateNaissance et Age). Si la classe Personne avait des méthodes, la classe Conducteur en hériterait de la même façon.

4. Classes abstraites

Classe abstraite
Une classe abstraite est une classe non instanciable. Elle exprime donc une généralisation abstraite, qui ne correspond à aucun objet existant du monde.

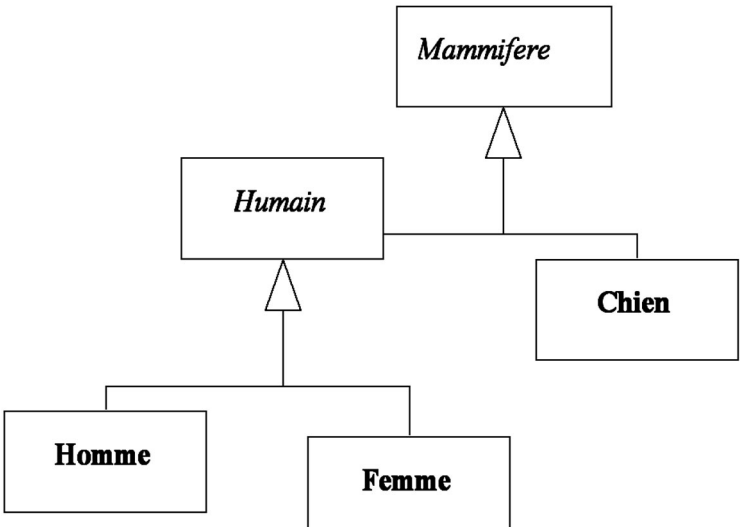
Classe abstraite et héritage
Une classe abstraite est *toujours héritée*. En effet sa fonction étant de généraliser, elle n'a de sens que si des classes en héritent. Une classe abstraite peut être héritée par d'autres classes abstraites, mais en fin de chaîne des classes non abstraites doivent être présentes pour que la généralisation est un sens.

Syntaxe : Notation d'une classe abstraite en UML
On note les classes abstraites en italique.



▲ IMG. 15 : NOTATION D'UNE CLASSE ABSTRAITE EN UML

Exemple de classe abstraite



▲ IMG. 16 : DES CHIENS ET DES HOMMES

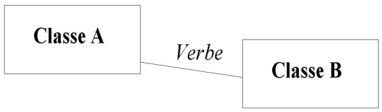
Dans la représentation précédente on a posé que les hommes, les femmes et les chiens étaient des objets instanciables, généralisés respectivement par les classes mammifère et humain, et mammifère.
Selon cette représentation, il ne peut donc exister de mammifères qui ne soient ni des hommes, ni des femmes ni des chiens, ni d'humains qui ne soient ni des hommes ni des femmes.

5. Association

Association

Une association est une relation logique entre deux classes (association binaire) ou plus (association n-aire) qui définit un ensemble de liens entre les objets de ces classes.
Une association est nommée, généralement par un verbe. Une association peut avoir des propriétés (à l'instar d'une classe). Une association définit le nombre minimum et maximum d'instances autorisée dans la relation (on parle de cardinalité).

Syntaxe

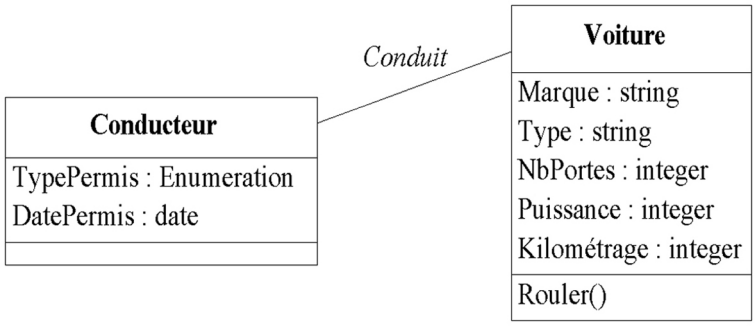


▲ IMG. 17 : NOTATION DE L'ASSOCIATION EN UML

Remarque

Une association est généralement bidirectionnelle (c'est à dire qu'elle peut se lire dans les deux sens). Les associations qui ne respectent pas cette propriété sont dites unidirectionnelles ou à navigation restreinte.

Exemple : L'association Conduit



▲ IMG. 18 : REPRÉSENTATION D'ASSOCIATION EN UML

L'association Conduit entre les classes Conducteur et Voiture exprime que les conducteurs conduisent des voitures.

6. Cardinalité

Cardinalité d'une association

La cardinalité d'une association permet de représenter le nombre minimum et maximum d'instances qui sont autorisées à participer à la relation. La cardinalité est définie pour les deux sens de la relation.

Syntaxe

Si min_a (resp. max_a) est le nombre minimum (resp. maximum) d'instances de la classe A autorisées à participer à l'association, on note sur la relation, à côté de la classe A : $min_a..max_a$.
Si le nombre maximum est indéterminé, on note n ou *.

Attention

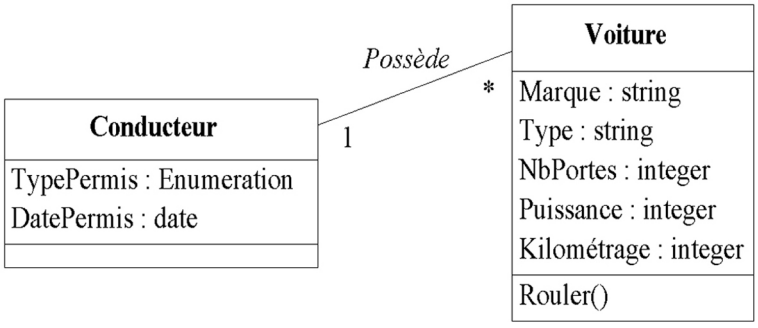
La notation de la cardinalité en UML est opposée à celle adopté en E-A. En UML on note à gauche (resp. à droite) le nombre d'instances de la classe de gauche (resp. de droite) autorisées dans l'association. En E-A, on note à gauche (resp. à droite) le nombre d'instances de la classe de droite (resp. de gauche) autorisées dans l'association.

Remarque

Les cardinalité les plus courantes sont :

- 0..1 (optionnel)
- 1..1 ou 1 (un)
- 0..n ou 0..* ou * (plusieurs)
- 1..n ou 1..* (obligatoire)

Exemple : La cardinalité de l'association Possède



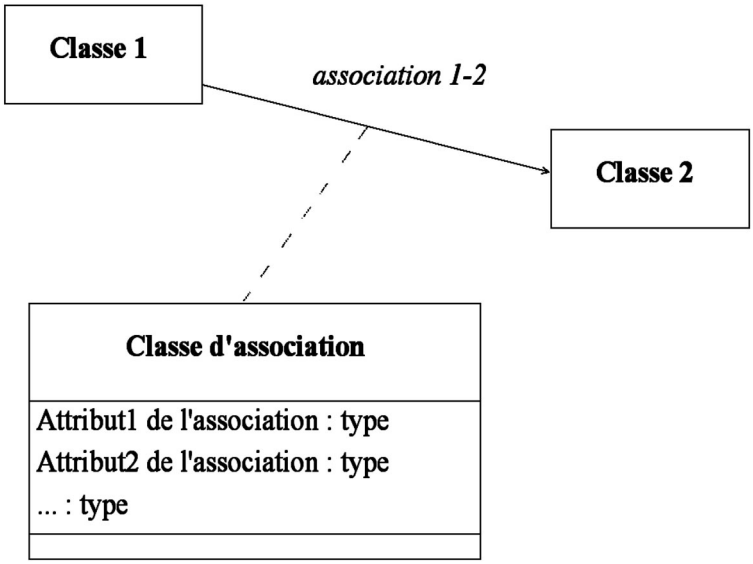
▲ IMG. 19 : REPRÉSENTATION DE CARDINALITÉ EN UML

Ici un conducteur peut possèder plusieurs voitures (y compris aucune) et une voiture n'est possédée que par un seul conducteur.

7. Classe d'association

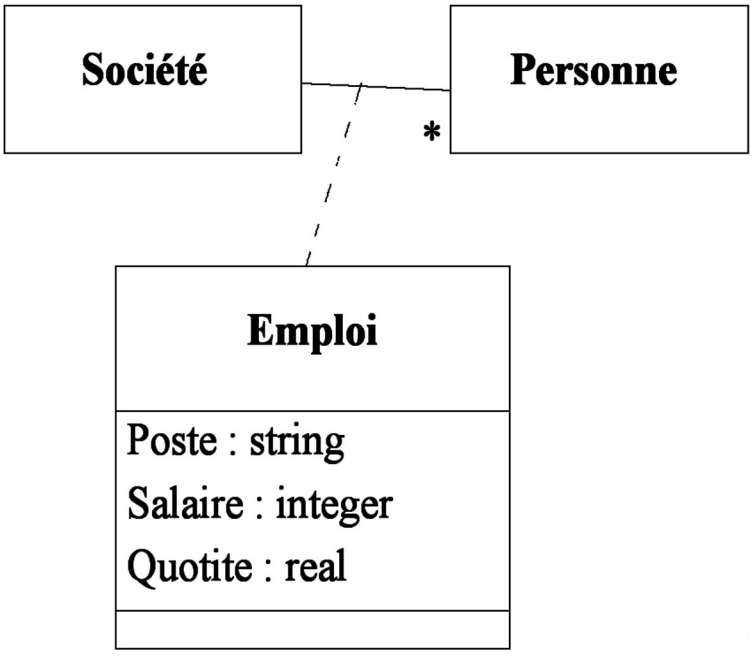
Association de composition
On utilise la notation des classes d'association lorsque l'on souhaite ajouter des propriétés à une association.

Syntaxe : Notation d'une classe d'association en UML



▲ IMG. 20 : NOTATION D'UNE CLASSE D'ASSOCIATION EN UML

Exemple de classe d'association



▲ IMG. 21 : EMPLOIS

8. Composition

Association de composition

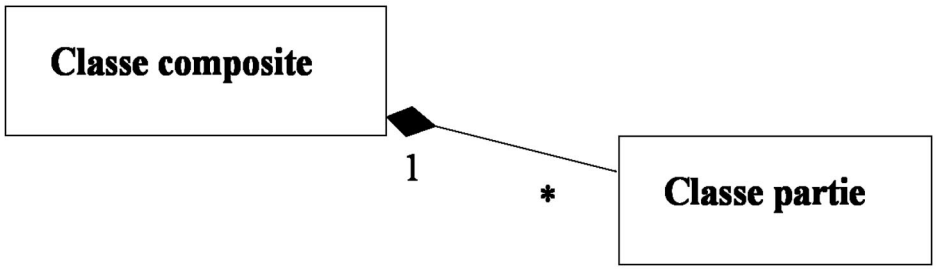
On appelle composition une association particulière qui possède les propriétés suivantes :

- La composition associe une classe composite et des classes parties, tel que tout objet partie appartient à un et un seul objet composite. C'est donc une association 1:N.
- La composition n'est pas partageable, donc un objet partie ne peut appartenir qu'à un seul objet composite à la fois.
- Le cycle de vie des objets parties est lié à celui de l'objet composite, donc un objet partie disparaît quand l'objet composite auquel il est associé disparait.

Remarque

La composition est une association particulière.

Syntaxe : Notation d'une composition en UML



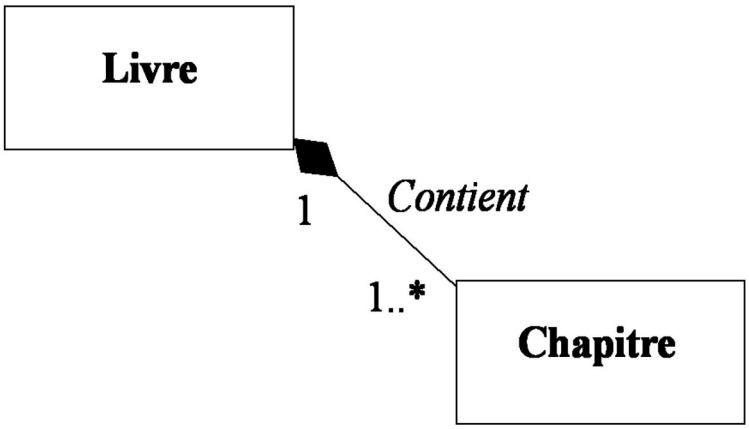
▲ IMG. 22 : NOTATION D'UNE COMPOSITION EN UML

Composition et cardinalité

La cardinalité côté composite est toujours de exactement 1.

Côté partie la cardinalité est libre, elle peut être 0..1, 1, * ou bien 1..*.

Exemple de composition



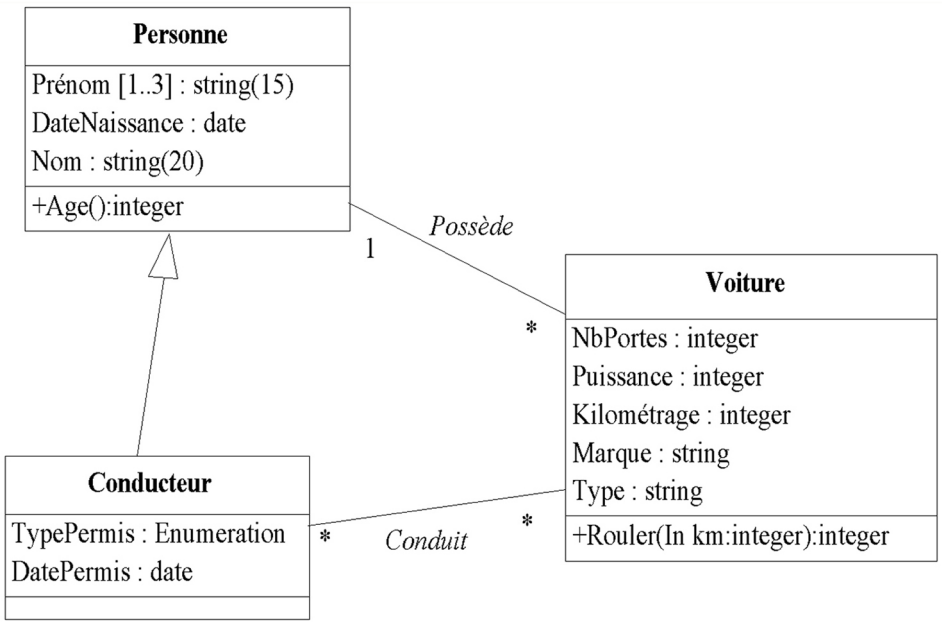
▲ IMG. 23 : UN LIVRE

On voit bien ici qu'un chapitre n'a de sens que faisant partie d'un livre, qu'il ne peut exister dans deux livres différents et que si le livre n'existe plus, les chapitres le composant non plus.

Remarque : Composition et entités faibles
La composition permet d'exprimer une association analogue à celle qui relie une entité faible à une entité identifiante en modélisation E-A. L'entité de type faible correspond à un objet partie et l'entité identifiante à un objet composite.

9. Des voitures et des conducteurs

Exemple



▲ IMG. 24 : EXEMPLE TRÈS SIMPLE DE DIAGRAMME DE CLASSES

Les relations de ce diagramme expriment que les conducteurs sont des personnes qui ont un permis ; que toute voiture est possédée par une unique personne (qui peut en posséder plusieurs) ; que les voitures peuvent être conduites par des conducteurs et que les conducteurs peuvent conduire plusieurs voitures.

Remarque
Les mots clés in, out et in/out devant un paramètre de méthode permettent de spécifier si le paramètre est une donnée d'entrée, de sortie, ou bien les deux.

Remarque
Le but d'une modélisation UML n'est pas de représenter la réalité dans l'absolu, mais plutôt de proposer une vision d'une situation réduite aux éléments nécessaires pour répondre au problème posé. Donc une modélisation s'inscrit toujours dans un contexte, et en cela l'exemple précédent reste limité car son contexte d'application est indéfini.

Pour aller plus loin...

"uml.free.fr", UML en Français, septembre, 2002.

Une très bonne référence en ligne sur la modélisation UML, avec des cours, des liens vers la norme, etc.

Le contenu dépasse très largement l'usage d'UML pour la modélisation de BD (et ne fait d'ailleurs pas de référence précise à ce sous-ensemble particulier).
On pourra consulter en particulier le chapitre sur les diagrammes de classe : http://uml.free.fr/cours/i-p14.html.

"http://developpeur.journaldunet.com/tutoriel/cpt/031013cpt_uml5conseils.shtml", Cinq petits conseils pour un

schéma UML efficace, **BORDERIE X,** avril, 2004.

Un conseil, c'est toujours bon à prendre !

"http://eric.univ-lyon2.fr/~jdarmont/docs/sise-bd-ex_uml.pdf", Exercices - Modèle UML, **DARMONT J,** janvier, 2004.

Des exercices corrigés sur la modélisation UML.

"http://www.info.uqam.ca/~godin/DiaposParChap/ChapI-5.ppt", Introduction au modèle relationnel, **GODIN R,** avril, 2004.

Un rappel du modèle logique relationnel, et du passage d'un modèle conceptuel UML au relationnel.

"http://developpeur.journaldunet.com/dossiers/alg_uml.shtml", UML en 5 étapes, **MORLON J,** avril, 2004.

Un tutoriel en ligne sur la modélisation UML. On consultera en particulier le tutoriel sur les diagrammes de classe : http://developpeur.journaldunet.com/tutoriel/cpt/010607cpt_umlintro.shtml

MULLER P, "Modélisation objet avec UML", Eyrolles, Paris, 1998.

ROQUES P, VALLÉE F, "UML 2 en action : De l'analyse des besoins à la conception J2EE", 385 pages, architecte logiciel, Eyrolles, Paris, 2004.

Pour un aperçu plus détaillé des possibilités d'expression du diagramme de classe UML, lire le chapitre 7 : Développement du modèle statique (pages 133 à 163).

On pourra notamment y trouver :

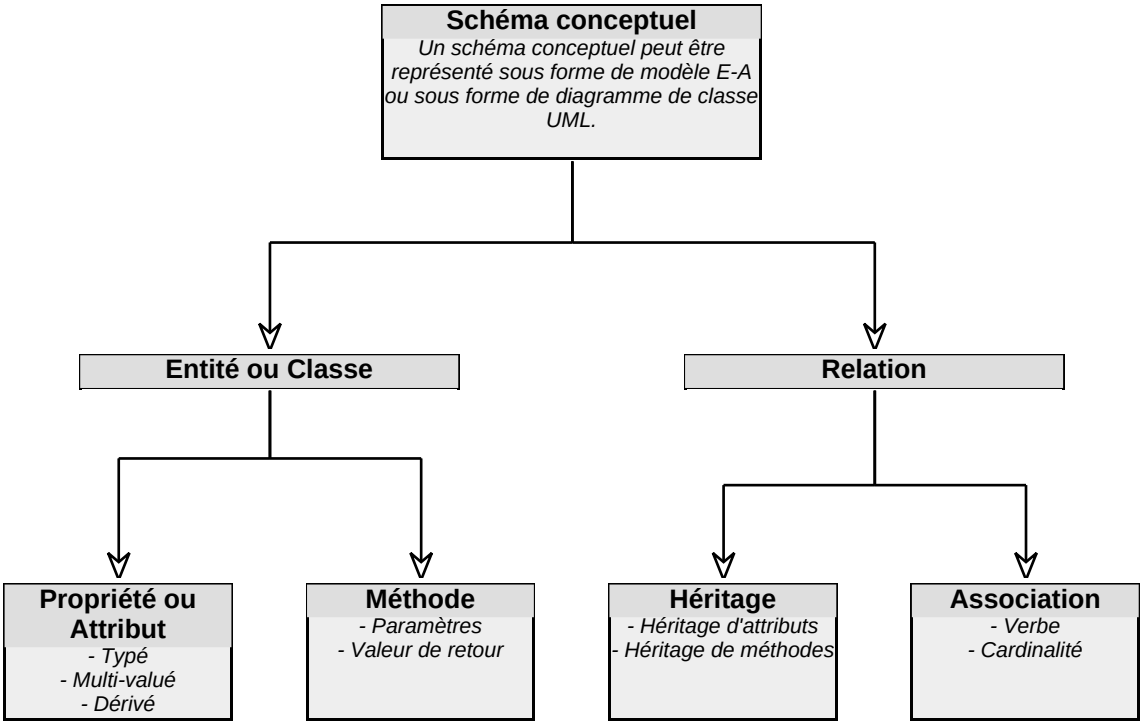
- L'association d'aggrégation
- Les propriétés d'association
- L'expression de rôles dans les associations
- Les attributs de classe
- Les qualificatifs
- Les opérations (ou méthodes)

Le chapitre donne de plus des conseils méthodologiques pour la conception (voir en particulier la synthèse page 163).
On pourra également y trouver :

- Des principes de choix de modélisation entre attributs et classes et sur la segmentation des classes
- Des principes de sélection des attributs (redondance avec les associations, avec les classes, etc.)
- Des principes de sélection des associations
- Des principes de choix de cardinalité (notamment pour la gestion d'historisation)
- Des principes de sélection des relations de généralisation (héritage)
- Des principes d'introduction de métaclasses (type)s

SOUTOU C, "De UML à SQL : Conception de bases de données", Eyrolles, 2002.

En résumé...



Chapitre C. Questions/réponses sur la modélisation

Question/Réponse 1. UML et E-A

Est ce qu'il y a des règles particulières pour le passage E-A à UML ?

Il n'y a pas de règle de passage UML à E-A car normalement on fait l'un ou l'autre. Mais les deux modèles sont effectivement équivalents, à part quelques détails (méthodes, classes abstraites, etc.).

Question/Réponse 2. Contraintes dynamiques

Comment représente-t-on les contraintes dynamiques sur les schémas conceptuels ? Par exemple si l'on veut dire que les deux clés étrangères "gagnant" et "perdant" d'une classe match (pointant sur des enregistrements d'une classe joueur) doivent être différents ?

En UML on représente les contraintes par une annotation sur le schéma, en E-A, on pose généralement ces contraintes dans un petit tableau après le schéma.

En SQL on verra qu'il existe une clause CHECK qui permet de gérer la plupart des contraintes simples. Sinon on devra faire appel aux langages associés à la BD (triggers PL/SQL typiquement sous Oracle, procédure VBA sous Access), voire s'en remettre à la couche applicative.

Cours général

II

Relationnel

Chapitre A. Modèle relationnel

Objectifs pédagogiques

Connaître les fondements du modèle relationnel.
Savoir effectuer le passage d'un schéma conceptuel à un schéma relationnel.

Section A1. Description du modèle relationnel

Le niveau logique

Le niveau logique est le lien entre le niveau conceptuel et l'implémentation effective de l'application. Le modèle conceptuel étant un modèle formel, le modèle logique a pour vocation d'être également un modèle formel, mais spécifiant non plus la réalité existante ou recherchée comme le modèle conceptuel, mais les données telles qu'elles vont exister dans l'application informatique.

Pour assumer cette fonction, le modèle relationnel s'est imposé (Codd, 1970) , en réaction aux insuffisances des modèles précédents, les modèles hiérarchique et réseau, et de part la puissance de ses fondements mathématiques.
Encore aujourd'hui dominant le modèle relationnel est un fondement indispensable à la conception de bases de données. De plus le modèle émergeant actuellement est le modèle relationnel-objet, et ce dernier est bien une extension du modèle relationnel qui le renforce et s'y appuie.

1. Le modèle relationnel

Le modèle relationnel a été introduit par Codd, en 1970 au laboratoire de recherche d'IBM de San José (Codd, 1970) . Il s'agit d'un modèle simple et puissant à la base de la majorité des bases de données aujourd'hui.

Modèle relationnel
Ensemble de concepts pour formaliser logiquement la description d'articles de fichiers plats, indépendamment de la façon dont ils sont physiquement stockés dans une mémoire numérique.
Le modèle relationnel inclut des concepts pour la *description* de données, ainsi que des concepts pour la *manipulation* de données.

Les objectifs du modèle relationnel, formulés par Codd, sont les suivants :
- Assurer l'indépendance des applications et de la représentation interne des données
- Gérer les problèmes de cohérence et de redondance des données
- Utiliser des langages de données basés sur des théories solides

Remarque : Extension du modèle relationnel
Le modèle relationnel est un standard, normalisé par l'ISO à travers son langage, le SQL. Il se veut néanmoins dès l'origine extensible, pour permettre de gérer des données plus complexes que les données tabulaires. Le modèle relationnel-objet est né de cette extension.

2. Domaine

Domaine
Ensemble, caractérisé par un nom, dans lequel des données peuvent prendre leurs valeurs.

Remarque
Un domaine peut-être défini en intention (c'est à dire en définissant une propriété caractéristique des valeurs du domaine) ou en extension (c'est à dire en énumérant toutes les valeurs du domaine)

Exemple : Domaines définis en intention
- Entier
- Réel
- Booléen
- Chaîne de caractères
- Monétaire : réel avec deux chiffres après la virgule
- Date : chaîne de 10 caractères comprenant des chiffres et des tirets selon le patron "00-00-0000"
- Salaire : Monétaire compris entre 15.000 et 100.000

Exemple : Domaines définis en extension
- Couleur : {Bleu, Vert, Rouge, Jaune, Blanc, Noir}
- SGBD : {Hiérarchique, Réseau, Relationnel, Objet, Relationnel-Objet}

3. Produit cartésien

Produit cartésien
Le produit cartésien, noté "X", des domaines D1, D2, ... , Dn, noté "D1 X D2 X ... X Dn" est l'ensemble des tuples (ou n-uplets ou vecteurs) <V1,V2,...,Vn> tel que Vi est une valeur de Di et tel que toutes les combinaisons de valeurs possibles sont exprimées.

Exemple

```
D1 = {A, B, C}
D2 = {1, 2, 3}
D1 X D2 = {<A,1>, <A,2>, <A,3>, <B,1>, <B,2>, <B,3>, <C,1>, <C,2>, <C,3>,}
```

4. Relation

Relation
Une relation sur les domaines D1, D2, ..., Dn est un sous-ensemble du produit cartésien "D1 X D2 X ... X Dn". Une relation est caractérisée par un nom.

- *Synonyme : Table.*

Syntaxe

On peut représenter la relation R sur les domaine D1, ... , Dn par une table comportant une colonne pour chaque domaine et une ligne pour chaque tuple de la relation.

D1	...	Dn
V1	...	Vn
...	...	...
V1	...	Vn

▲ TAB. 1 : RELATION R

Remarque

Une relation est toujours définie en extension, par l'énumération des tuples la composant.

5. Attribut et enregistrement

Attribut

On appelle attribut d'une relation, une colonne de cette relation. Un attribut est caractérisé par un nom et un domaine dans lequel il prend ses valeurs.

- *Synonymes : Champs, Propriété, Colonne.*

Enregistrement

On appelle enregistrement d'une relation, une ligne de cette relation. Un enregistrement prend une valeur pour chaque attribut de la relation.

- *Synonymes : Tuple, N-uplet, Vecteur, Ligne.*

Exemple

A	B
1	1
1	2
2	2

▲ TAB. 2 : RELATION R

La relation R comporte les deux attributs A et B et les trois enregistrements <1,1>, <1,2> et <2,2>

Remarque : Attribut, domaine, ordre

Un attribut se distingue d'un domaine car il peut ne comporter que certaines valeurs de ce domaine.

Les colonnes de la relation ne sont pas ordonnées et elles ne sont donc repérées que par le nom de l'attribut.

Remarque : Valeur nulle

Un enregistrement peut ne pas avoir de valeur pour certains attributs de la relation, parce que cette valeur est inconnue ou inapplicable, sa valeur est alors "null".

6. La relation Vol

Exemple

Numero	Compagnie	Avion	Départ	Arrivée	Date
AF3245	Air France	747	Paris	Oulan Bator	01-08-2002
AF6767	Air France	A320	Paris	Toulouse	30-07-2002
KLM234	KML	727	Paris	Amsterdam	31-07-2002

▲ TAB. 3 : RELATION VOL

7. Clé

Clé
Groupe d'attributs minimum qui détermine un tuple unique dans une relation.

Attributs de clés toujours valués
Afin d'être déterminants pour l'identification d'un enregistrement, tous les attributs d'une clé doivent être valués, c'est à dire qu'aucun ne peut avoir de valeur "null".

Remarque : Clé primaire
Toute relation doit comporter au moins une clé, ce qui implique qu'une relation ne peut contenir deux tuples identiques.
Si plusieurs clés existent dans une relation, on en choisit une parmi celles-ci. Cette clé est appelée *clé primaire*. La clé primaire est généralement choisie de façon à ce qu'elle soit la plus simple, c'est à dire portant sur le moins d'attributs et sur les attributs de domaine les plus basiques (entiers ou chaînes courtes typiquement).

Remarque : Détermination d'une clé
Définir un groupe d'attributs comme étant une clé nécessite une réflexion sémantique sur les données composant ces attributs, afin de s'assurer de leur unicité.

Exemple

- L'attribut numéro de sécurité sociale d'une relation personne est une bonne clé car son unicité est assurée sémantiquement.
- Le groupe d'attributs nom, prénom d'une relation personne est en général une mauvaise clé, car les homonymes existent.

Remarque : Clé artificielle
S'il est impossible de trouver une clé primaire, ou que les clés candidates sont trop complexes, il est possible de faire appel à une clé artificielle. Une clé artificielle est une propriété supplémentaire ajoutée au schéma de la relation, qui n'est reliée à aucune signification, et qui sert uniquement à identifier de façon unique les enregistrements et/ou à simplifier les références de clés étrangères.

Remarque : Clé artificielle et niveau logique
Au niveau du modèle logique, il faut éviter la simplicité consistant à identifier toutes les relations avec des clés artificielles, et ne réserver cet usage qu'aux cas particuliers.

Remarque : Clé artificielle et niveau physique, évolutivité, maintenance et performance
Au niveau de l'implémentation physique par contre, il est courant que des clés artificielles soient utilisées de façon systématique.
Du point de vue de *l'évolutivité* de la BD, il existe toujours un risque qu'une clé non-artificielle perde sa propriété d'unicité ou de non-nullité.
Du point de vue de *la maintenance* de la BD, il existe toujours un risque qu'une clé non-artificielle voit sa valeur modifiée et dans ce cas, la répercution de ce changement pour mettre à jour toutes les références peut poser problème.
Du point de vue de *la performance* de la BD, les clés non-artificielles ne sont pas en général optimisées en terme de type et de taille, et donc peuvent limiter les performances dans le cadre des jointures.

Précisons néanmoins qu'inversement les clés artificielles ont pour conséquence de systématiser des jointures qui auraient pu être évitées avec des clés primaires signifiantes.

Exemple : Problème d'évolutivité posé par une clé signifiante
Soit le numéro de sécurité sociale la clé primaire d'une table d'une BD française, elle ne permettra pas d'entrer un individu non-français issu d'un pays ne disposant pas d'un tel numéro.

Exemple : Problème de maintenance posé par une clé signifiante
Soit le numéro de sécurité sociale la clé primaire d'une table d'une BD centrale dont les données sont exploitées par d'autres tables d'autres BD qui viennent "piocher" dans cette BD pour leurs propres usages, sans que la BD centrale ne connaisse ses "clients". Soit une erreur dans la saisie d'un numéro de sécurité sociale dans la BD centrale, si ce numéro est corrigé, il faudrait (ce qui n'est pas possible dans notre cas) impérativement en avertir toutes les bases utilisatrices pour qu'elles mettent à jour leurs références.

Exemple : Problème de performance posé par une clé signifiante
Soit le numéro de sécurité sociale la clé primaire d'une table comptant un million d'enregistrements, ce numéro est généralement un nombre à 13 chiffres ou une chaîne à 13 caractères, ce qui dans les deux cas est supérieur au nombre à 7 chiffres suffisant pour identifier tous les individus de la BD. Les performances seront donc toujours moins bonnes, lors des jointures, si une clé prend deux fois plus de place en mémoire que son optimum. Mais ajoutons que cette perte de performance n'a pas toujours de conséquence sur la réalité perceptible par les utilisateurs de la BD.
Inversement, soit une clé artificielle la clé primaire d'une table T1, par ailleurs référencée par une autre table T2. Soit le numéro de sécurité sociale un attribut clé de T1. Si l'on veut par requête disposer des infos de T2 ainsi que du numéro de sécurité sociale de T1, alors il faudra faire une jointure, tandis que si ce numéro signifiant avait été choisi comme clé primaire, cela n'aurait pas été nécessaire.

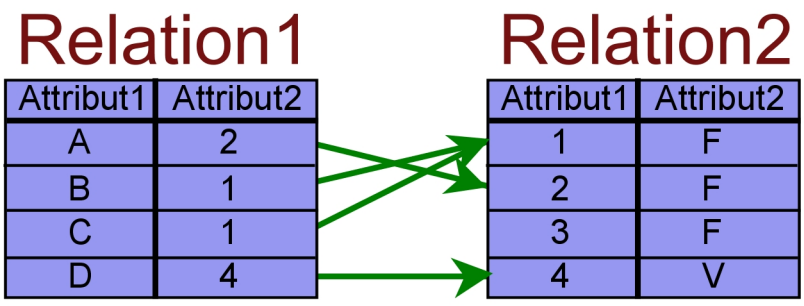
8. Lien entre relations

Le modèle relationnel a pour objectif la structuration de données selon des relations. L'enjeu est de parvenir à traduire un modèle conceptuel en modèle logique relationnel. Typiquement si le formalisme conceptuel utilisé est le modèle E-A, il faudra pouvoir traduire les entités et les associations en relations. Afin que la représentation des données ne soit pas redondante, il est nécessaire que les données soit réparties dans de multiples relations et non regroupées dans une seule. Or un modèle conceptuel informe sur les liens entre les entités et associations, cela est traduit graphiquement par les lignes qui les relient. Il est donc fondamental que le modèle relationnel puisse également maintenir cette information, à savoir les liens entre les données réparties dans les relations.

Afin de représenter des liens entre relations dans un modèle relationnel, la seule solution est de stocker l'information dans une relation, et donc que certains attributs d'une relation servent à pointer sur d'autres relations.

Lien
Le lien entre deux tuples A=>B de deux relations différentes est matérialisable par une référence depuis l'un des tuples, A, à la clé primaire de l'autre tuple, B.

Exemple



▲ SCH. 4 : ILLUSTRATION

L'attribut "Attribut2" de la relation "Relation1" référence l'attribut "Attribut1" de la relation "Relation2" ("Attribut1" est la clé primaire de "Relation2").

Remarque : Direction du lien
Dans un modèle relationnel, un lien est toujours unidirectionnel.

9. Eclatement des relations pour éviter la redondance

Relation redondante

Numero	Date	Gare1	Gare2	Train	Vitesse	Nom	Prénom
1010	01-01-2001	Paris	Lyon	TGV	450	Dupont	Joëlle
1011	02-01-2001	Paris	Limoges	TER	200	Durand	Jean-Pierre
1012	03-01-2001	Paris	Madrid	TGV	450	Dupont	Joëlle
1013	03-01-2001	Lyon	Limoges	TER	200	Dupont	Jean-Pierre

▲ TAB. 4 : RELATION VOYAGEENTRAIN

Dans la représentation précédente, les voyages en train sont représentés dans une unique relation, qui contient des informations relatives au voyage lui même (numéro, date, départ, arrivée), mais aussi au train utilisé pour le voyage (type de train et vitesse maximale), et au conducteur du train (nom et prénom).

Cette représentation, bien que très simplifiée par rapport à la réalité (on imagine facilement plusieurs dizaines d'attibuts possibles) est redondante. En effet chaque fois qu'un voyage mobilisera un train TGV, la vitesse maximale de 450 km/h devra aussi être rappelée. De même pour chaque conducteur, il faudra rappeler le nom et le prénom.
Cette redondance pose un certain nombre de problèmes :

- **Incohérence**

  Imaginons qu'une faute de saisie se glisse dans l'orthographe du nom de Dupont (un "d" à la place du "t") pour le voyage 1010, il sera impossible de savoir que c'est la même personne qui conduit le train pour le voyage 1012 (car Joëlle Dupond peut exister et être conductrice de train).

- **Mise à jour**

  Imaginons que Joëlle Dupont se marie et change de nom, il faudra changer cette information pour tous les voyages auxquels participe cette conductrice. Ceci est également un risque d'erreur, qui renvoie au risque d'incohérence précédemment mis en exergue.

- **Perte d'information**

  Imaginons que temporairement plus aucun voyage n'existe pour des TGV. Tous les enregistrements portant sur les TGV disparaitront. On perdra alors l'information comme quoi la vitesse d'un TGV est de 450 km/h, car cette information n'existera plus dans la base. Or l'information intrinsèque au TGV, qui est sa vitesse maximale, n'est pas liée à un voyage en particulier, et donc il est dommage de ne pas conserver cette information indépendamment du fait que les TGV sont ou non utilisés à un instant t.

- **Dépendance des insertions**

  Imaginons que nous souhaitions représenter dans la base de données un nouveau conducteur, qui n'est encore affecté à aucun voyage. Il est impossible d'ajouter une telle information, car l'insertion d'une personne est dépendant de l'insertion d'un tuple complet portant également sur un voyage. Il est évidemment très mauvais d'imaginer des solutions de contournement, telles que par exemple un tuple avec des valeurs nulles sur toutes les propriétés sauf les nom et prénom.

Relation éclatée

Le bon usage du modèle relationnel consiste donc à éclater les informations dans de multiples relations, afin de ne pas conserver de redondance. Dans le cas précédent, on préfèrera donc un découpage de la relation VoyageEnTrain en trois relations, Voyage, Modele et Conducteur, chacune représentant des informations correspondant à des objets différents (notons l'ajout d'une clé artificielle numéro pour la nouvelle relation conducteur, non identifiable de façon unique par les attributs nom et prénom).

Numero	Date	Gare1	Gare2
1010	01-01-2001	Paris	Lyon
1011	02-01-2001	Paris	Limoges
1012	03-01-2001	Paris	Madrid
1013	03-01-2001	Lyon	Limoges

▲ TAB. 5 : RELATION VOYAGE SANS LIEN

Modele	Vitesse
TGV	450
TER	200

▲ TAB. 6 : RELATION MODELE

Numero	Nom	Prénom
1	Dupont	Joëlle
2	Durand	Jean-Pierre

▲ TAB. 7 : RELATION CONDUCTEUR

Relation éclatée avec liens

Dans cette représentation éclatée des données précédemment regroupées dans la même relation, on a perdu des informations importantes, relatives aux liens entre les voyages, les modèles de train et les conducteurs. En effet on ne sait plus que le voyage numéro 1010 s'effectue sur un TGV avec la conductrice Joëlle Dupont. Il est indispensable, pour conserver l'ensemble des informations, de matérialiser à nouveau ces liens, en ajoutant des références dans la relation Voyage, aux tuples de Modele et Conducteur.

Numero	Date	Gare1	Gare2	Modele	Conducteur
1010	01-01-2001	Paris	Lyon	TGV	1
1011	02-01-2001	Paris	Limoges	TER	2
1012	03-01-2001	Paris	Madrid	TGV	1
1013	03-01-2001	Lyon	Limoges	TER	2

▲ TAB. 8 : RELATION VOYAGE AVEC LIENS

10. Clé étrangère

Clé étrangère

Groupe d'attributs d'une relation R1 devant apparaître comme clé dans une autre relation R2 afin de matérialiser un lien entre les tuples de R1 et les tuples de R2. La clé étrangère d'un tuple référence la clé primaire d'un autre tuple.

Contrainte d'intégrité référentielle

Une clé étrangère respecte la contrainte d'intégrité référentielle si sa valeur est effectivement existante dans la clé primaire d'un tuple de la relation référencée, ou si sa valeur est "null".
Une clé étrangère qui ne respecte pas la contrainte d'intégrité référentielle exprime un lien vers un tuple qui n'existe pas et donc n'est pas cohérente.

Remarque : Suppression et mise à jour en cascade

Pour que la cohérence soit conservée, lors de la suppression d'un tuple référencé par d'autres tuples, les tuples référençants doivent également être supprimés. Sinon leur clé étrangère pointerait vers des tuples qui n'existent plus et la contrainte d'intégrité référentielle ne serait plus respectée. Afin de maintenir la cohérence, il faut donc procéder à une supression en cascade (sachant que les tuples référençant peuvent également être référencés par ailleurs).
La problématique est la même lors d'une mise à jour de la clé primaire d'un tuple : il faut alors mettre à jour toutes les clés étrangères référençant ce tuple.
Notons que certains SGBD peuvent se charger automatiquement de ces suppressions et mises à jour en cascade.

11. Schéma relationnel

Schéma d'une relation

Le schéma d'une relation définit cette relation par intention. Il est composé du nom de la relation, de la liste de ses attributs avec les domaines respectifs dans lesquels ils prennent leurs valeurs, de la clé primaire, et des clés étrangères.

Schéma relationnel d'une base de donnée

Le schéma relationnelle d'une BD est la définition en intention de cette BD (par opposition à l'instance de la BD qui est une extension de la BD). Il est composé de l'ensemble des schémas de chaque relation de la BD.

Syntaxe : Relation

```
Relation (Attribut1:Domaine1, Attribut2:Domaine2, ... , AttributN:DomaineN)
```

La relation "Relation" contient N attributs chacun défini sur son domaine.

Syntaxe : Clé primaire

```
Relation (Attribut1:Domaine1, ... , AttributM:DomaineM, ... , AttributN:DomaineN)
```

La clé de la relation "Relation" est composée des attributs "Attribut1" à "AttributM".
En général on note la clé primaire en premier dans la relation.

Syntaxe : Clé étrangère

```
Relation1 (..., AttributM=>Relation2, ... , AttributN=>Relation2)
```

La relation "Relation1" comporte une clé étrangère (composée des attributs "AttributM" à "AttributN") référençant la clé primaire de "Relation2". Bien sûr il peut exister plusieurs clés étrangères vers plusieurs relations distinctes. Une clé étrangère et sa clé primaire référencée sont toujours composées du même nombre d'attributs. Il n'est pas nécessaire de préciser les domaines des attributs appartenant à la clé étrangère car ce sont forcément les mêmes que ceux de la clé primaire référencée. Il n'est pas non plus en général nécessaire de préciser dans le schéma relationnel quels attributs de la clé étrangère référencent quels attributs de la clé primaire (cela est généralement évident) mais il est possible de la faire on notant "Attribut=>Relation.Attribut".
En général on note les clés étrangères en dernier dans la relation, sauf pour les clés étrangères qui font partie de la clé primaire (clés identifiantes).

Clés candidates

On ne représente pas, en général, sur le schéma relationnel les clés candidates afin de ne pas alourdir la notation. On recommande néanmoins de dresser la liste des clés candidates (non choisies comme clés primaires) en annotation du schéma relationnel afin d'en conserver la mémoire.
On peut néanmoins noter les clés candidates en les soulignant en pointillé.

12. Exemple de schéma relationnel simple pour la géographie

Exemple

```
Personne (Numero:Entier, Nom:Chaîne, Prénom:Chaîne, LieuNaissance=>Ville)
Pays (Nom:Chaîne, Population:Entier, Superficie:Réel, Dirigeant=>Personne)
Région (Pays=>Pays, Nom:Chaîne, Superficie, Dirigeant=>Personne)
Ville (CodePostal:CP, Nom:Chaîne, Pays=>Région.Pays, Région=>Région.Nom,
Dirigeant=>Personne)
```

Le schéma relationnel précédent décrit :

- **Des personnes**

  Elles sont identifiées par un numéro qui est en fait une clé artificielle. En effet, même une clé composée de tous les attributs (Nom, Prénom, LieuNaissance) laisse une possibilité de doublons (homonymes nés dans la même ville).

  La clé étrangère LieuNaissance fait référence à la relation Ville, et plus précisément à sa clé primaire CodePostal, ce qui est est laissé implicite car évident.

- **Des pays**

  Ils sont identifiés par leur nom, puisque deux pays ne peuvent avoir le même nom.

  Les pays sont dirigés par des personnes, et ce lien est matérialisé par la clé étrangère Dirigeant.

- **Des régions**

  Elles font partie d'un pays et ont un nom. Deux régions de pays différents pouvant avoir le même nom, il faut utiliser une clé primaire composée à la fois du nom de la région et du nom du pays, qui est une clé étrangère (le nom est appelé clé locale car il n'est pas suffisant pour identifier un tuple de la relation Région, et la clé étrangère vers la relation Pays est appelée clé identifiante).

- **Des villes**

  Elles sont identifié par un code postal qui est unique dans le monde (en utilisant le prefixe de pays de type "F-60200"). Ce code postal à pour domaine CP qui est une chaîne composée d'une ou deux lettres, d'un tiret, puis d'une série de chiffres.

  Le lien d'appartenance entre une ville et une région est matérialisé par la clé étrangère composée des deux attributs Pays et Région. Cette clé référence la clé primaire de la relation Région, également composée de deux attributs. Pour clairement expliciter les références (bien que sémantiquement la dénomination des attributs ne laisse pas de place au doute) on utilise la syntaxe Région.Pays et Région.Nom.

Section A2. Le passage E-A vers Relationnel

Afin de pouvoir implémenter une base de données, il faut pouvoir traduire le modèle conceptuel en modèle logique. Cela signifie qu'il faut pouvoir convertir un modèle E-A ou un modèle UML en modèle relationnel. Dans les deux cas, E-A ou UML, les modèles conceptuels sont suffisamment formels pour ce passage soit systématisé. Nous étudions à présent les règles permettant de réaliser ce passage, en nous concentrant sur le passage du modèle E-A au modèle relationnel. Les règles de passage de UML au modèle relationnel seront suffisamment similaires pour ne pas être détaillées.

1. Transformation des entités

Entité identifiée

Pour chaque entité (identifiée) E, on crée une relation R dont le schéma est celui de l'entité. La clé primaire de R est une des clés de E.

Entité non identifiée

Pour chaque entité non identifiée I ayant un identifiant étranger E, on crée une relation R qui comprend tous les attributs de I, ainsi que les attributs clés de la relation correspondant à E. La clé de R est la concaténation de la clé locale de I et de la clé de E.

2. Transformation des associations

Association 1:N

Pour chaque association binaire A de type 1:N (le cas échéant 0,1:N) entre les entités S et T (représentés par les relations RS et RT respectivement) on inclut dans la définition de RT comme clé étrangère la clé de RS ainsi que tous les attributs simples de A.

Association M:N et associations de degré supérieur à 2

Pour chaque association binaire A de type M:N ou pour chaque association A de degré supérieur à 2, on crée une nouvelle relation RA pour représenter A. On met dans RA comme clé étrangère, les clés de toutes les relations correspondant aux entités participant à A et dont la concaténation formera sa clé. On ajoute également à RA (et éventuellement dans sa clé pour les attributs clés) les attributs définis sur A.

Association 1:1

Pour chaque association binaire A de type 1,1:1,1 ou 1,1:1,0 entre les entités S et T (représentées par les relations RS et RT respectivement) on substitut dans la définition de RS l'éventuelle clé primaire (si RS était identifiée) par la clé primaire de RT qui est également définie comme clé étrangère vers RT. On ajoute également à RS l'ensemble des attributs de A. Notons que dans le cas d'une association 1,1:1,1 RT peut-être choisie à la place de RS pour accueillir la clé étrangère.
Pour chaque association binaire A de type 0,1:0,1 entre les entités S et T (représentées par les relations RS et RT respectivement) on inclut dans la définition de RS comme clé étrangère la clé de RT ainsi que tous les attributs simples de A. La clé étrangère vers RT incluse dans RS est également définie comme étant unique, afin d'assurer la cardinalité maximum de 1. Notons qu'il n'était pas possible dans ce cas 0,1:0,1 que la clé étrangère vers RT incluse dans RS soit également la clé primaire, car la cardinalité de

0 supposerait que la clé primaire puisse être "null", ce qui est illégal pour une clé.

Remarque : Cardinalité minimale 0 ou 1
Selon que la cardinalité minimale est 0 ou 1 (ou plus) du côté de la relation référençante on ajoutera ou non une contrainte de non nullité sur la clé étrangère.
Selon que la cardinalité minimale est 0 ou 1 (ou plus) du côté de la relation référencée on ajoutera ou non une contrainte d'existance de tuples référençant pour chaque tuple de la relation référencée.

3. Remarques concernant la transformation des associations 1:1

Remarque : Choix de la relation référençante pour la traduction de l'association 1:1
La traduction d'une association 1,1:1,1 ou 0,1:0,1 conduit à devoir faire un choix entre les deux entités composant l'association, afin de déterminer laquelle est référencée de laquelle est référençante[Dans le cas d'une association 0,1:1,1, c'est toujours l'entité coté 1,1 qui sera choisie pour référencer.]. Ce choix peut-être arbitraire, mais peut également être éclairé par la sémantique des relations, et l'on choisira alors celle qui est de plus faible importance du point de vue de la modélisation pour référencer (et donc intégrer la clé étrangère).

Exemple de choix de la relation référençante pour une relation 1:1
Soit deux entités, "homme" et "femme" et une association "mariage" de cardinalité 1,1:1,1 (hommes et femmes sont donc obligatoirement mariés) unissant ces deux entités. Les entités "homme" et "femme" sont identifiées par un attribut "nom". S'il serait plus galant de choisir "femme" comme entité principale, il sera dans la pratique plus opportun de choisir "homme", car le "nom" de l'entité "homme" sera effectivement une propriété pertinente pour l'entité "femme" et donc aura plus de sens en tant que clé primaire et en tant que clé étrangère (dans le cas inverse, la clé de l'homme serait le nom de sa femme, ce qui ne correspond pas à la réalité administrative du mariage). On choisira donc plutôt la représentation :

```
homme (nom)
femme (nomMariage=>homme, nomJeuneFille) avec nomJeuneFille clé
```

Si l'association avait été de cardinalité 0,1:0,1 (certains hommes et femmes ne sont pas mariés), le même choix se serait imposé et l'on aurait abouti au résultat suivant:

```
homme (nom)
femme (nomJeuneFille, nomMariage=>homme) avec nomMariage unique
```

Remarque : Fusion des relations dans le cas de la traduction de l'association 1:1
Il est possible, dans le cas d'une association 1,1:1,1 de décider de fusionner les deux relations en une seule. La relation résultante est alors composée de l'ensemble des attributs de chacune des deux relations, en choisissant l'une des deux clés de l'une des deux relations pour identifier la nouvelle relation[C'est également possible pour les associations 0,1:1,1, mais c'est généralement moins pertinent, puisque cela conduira à un certain nombre de champs null. Ce n'est jamais pertinent dans le cas de relations 0,1:0,1.].
Ce choix sera conduit par une appréciation du rapport entre :

- La complexité introduite par le fait d'avoir deux relations là ou une suffit,
- La pertinence de la séparation des deux relations d'un point de vue sémantique,
- Les pertes de performance dues à l'éclatement des relations,
- Les pertes de performance dues au fait d'avoir une grande relation,
- Les questions de sécurité et de sureté factorisées ou non au niveau des deux relations,
- etc.

Dans le doute il est préférable d'adopter l'approche systématique consistant à séparer les deux relations.

4. Transformation des attributs

Attributs simples
Pour chaque attribut élémentaire et monovalué A d'une entité E (idem pour les associations), on crée un attribut correspondant à A sur la relation RE correspondant à E.

Attributs composites

Pour chaque attribut composite C, comprenant N sous-attributs, d'une entité E (idem pour les associations), on crée N attributs correspondants à C sur la relation RE correspondant à E.

Attributs multivalués

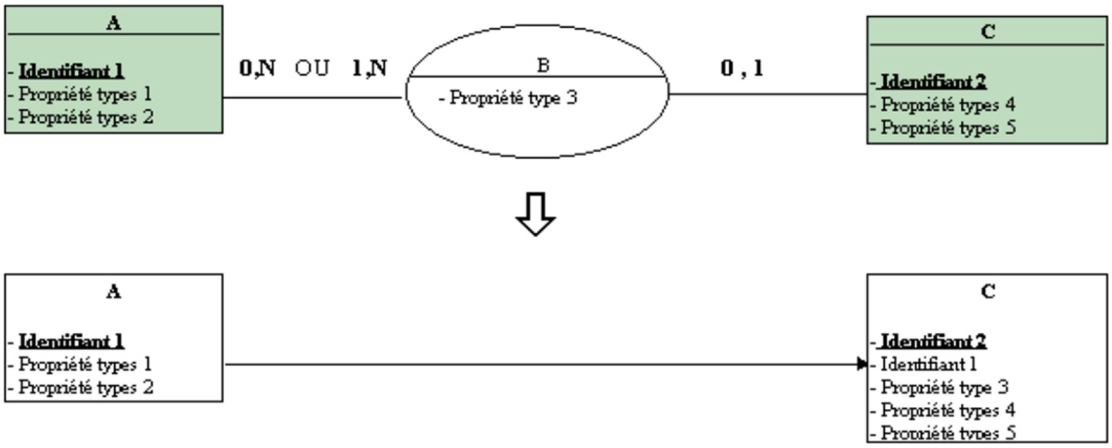
Pour chaque attribut multivalué M d'une entité E (idem pour les associations), on crée une nouvelle relation RM qui comprend un attribut monovalué correspondant à M ainsi qu'une clé étrangère vers RE (relation représentant E). La clé de RM est la concaténation des deux attributs.
On gère donc M comme s'il s'agissait d'une entité faible RM, dont la clé locale serait M, et qui serait associée à E par une relation 1:N.

Attributs dérivés

On ne représente pas en général les attributs dérivés dans le modèle relationnel, ils seront calculés dynamiquement soit par des procédures internes à la BD (procédures stockées), soit par des procédures au niveau applicatif.
On peut néanmoins décider (pour des raisons de performance essentiellement) de représenter l'attribut dérivé comme s'il s'agissait d'un attribut simple, mais il sera nécessaire dans ce cas d'ajouter des mécanismes de validation de contraintes dynamiques (avec des déclencheurs par exemple) pour assurer que la valeur stockée évolue en même temps que les attributs sur lesquels le calcul dérivé porte. Notons qu'introduire un attribut dérivé dans le modèle relationnel équivaut à introduire de la redondance, ce qui est en gérénal déconseillé, et ce qui doit être dans tous les cas contrôlé.

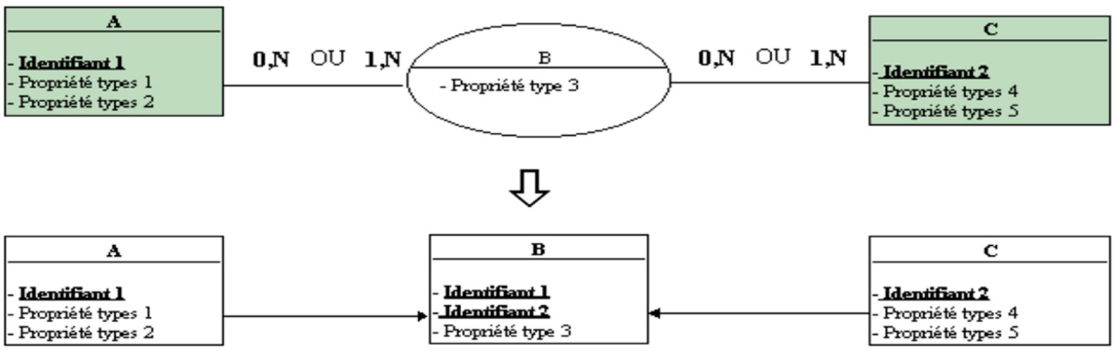
5. Exemple de passage E-A vers relationnel

Exemple



▲ IMG. 25 : TRANSFORMATION AVEC UNE RELATION 1:N

Exemple



▲ IMG. 26 : TRANSFORMATION AVEC UNE RELATION M:N

6. Transformation de la relation d'héritage

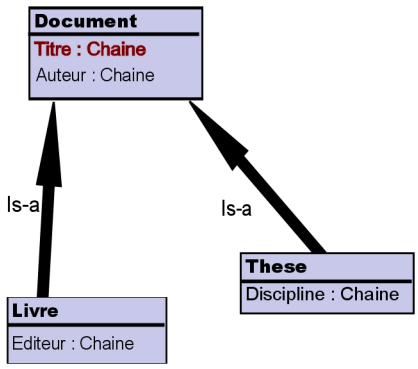
Le modèle relationnel ne permet pas de représenter directement une relation d'héritage, puisque que seuls les concepts de relation et de référence existent dans le modèle. Il faut donc appauvrir le modèle conceptuel pour qu'il puisse être représenté selon un schéma relationnel.

Trois solutions existent pour transformer une relation d'héritage :
- Représenter l'héritage par une référence entre la classe mère et la classe fille.
- Représenter uniquement les classes filles par des relations.
- Représenter uniquement la classe mère par une relation.

Les entités non finales sont abstraites
En modélisation E-A on considèrera toujours que les entités non finales (c'est à dire qui sont héritées par d'autres entités) sont *abstraites.*Une entité abstraite est une entité qui ne peut pas être instanciée.
Donc si E2 hérite de E1 (et que E2 est finale c'est à dire qu'aucune classe n'hérite de E2), il existera des objets de E2, mais pas des objets de E1. Si l'on veut disposer d'objets de E1, il suffit de créer une classe E1' qui hérite de E1 sans apporter de propriété supplémentaire.
En modélisation UML on pourra différencier les classes abstraites des classes instanciables.

7. Exemple de transformation d'une relation d'héritage

Exemple
Soit le modèle E-A suivant :



▲ SCH. 5 : REPRÉSENTATION DE DOCUMENTS

Il existe trois façons de traduire la relation d'héritage : par référence, par absorption par les sous-types d'entité, par absorption par l'entité générale.

Remarque : Type d'héritage
Si une thèse peut également être un livre (et dans la réalité c'est bien le cas puisqu'une thèse peut-être publiée), alors l'héritage n'est pas exclusif puisque un même document peut être une thèse *et* un livre.
L'héritage n'est pas non plus complet, puisque l'on observe que les thèses et les livres ont des attributs qui ne sont pas communs (la discipline pour la thèse et l'éditeur pour le livre).
En toute rigueur, il faudrait donc appliqué le cas général (ni exclusif, ni complet) et appliqué une traduction de l'héritage par référence. Observons néanmoins les avantages et inconvénients que chacun des trois choix porterait.

Héritage représenté par une référence

```
Document(Titre:Chaîne, Auteur:Chaîne)
These(Titre=>Document, Discipline:Chaîne)
Livre(Titre=>Document), Editeur:Chaîne
```

Remarque : Avantages du choix
Il n'y a ni redondance ni valeur nulle inutile dans les relations These et Livre, c'est la traduction la plus juste pour le problème posé.

Remarque : Inconvénient du choix
Il est relativement lourd de devoir créer un tuple dans Document pour chaque tuple dans These ou Livre, alors que la relation Document ne porte que l'information concernant l'auteur, juste pour assurer les cas, par ailleurs plutôt rares dans la réalité, où les thèses sont publiées sous forme de livres.

Héritage absorbé par les sous-types d'entité

```
These(Titre, Discipline:Chaîne, Auteur:Chaîne)
Livre(Titre), Editeur:Chaîne, Auteur:Chaîne
```

Remarque : Avantages du choix
Il est plus simple que le précédent, puisque la représentation d'une thèse ou d'un livre ne nécessite plus que d'ajouter un seul tuple. Il évite donc les clés étrangères, toujours plus délicates à gérer d'un point de vue applicatif.

Remarque : Inconvénient du choix
Lorsqu'une thèse est publiée sous la forme d'un livre, alors une information sera dupliquée (l'auteur), ce qui introduit de la redondance. Si une telle redondance peut-être tolérée pour des raisons de simplification ou de performance (si on considère que le cas est rare par exemple et donc que l'héritage est "presque" exclusif), il sera néanmoins nécessaire d'assurer son contrôle par un moyen supplémentaire.
Dans le cas général, il n'est pas souhaitable de tolérer une telle redondance.

Héritage absorbé par l'entité générale

```
Document(Titre, Discipline:Chaîne, Editeur:Chaîne, Auteur:Chaîne,
Type:{These|Livre|These+Livre})
```

Remarque : Avantages du choix
Il est plus simple que le premier, puisque la représentation d'une thèse ou d'un livre ne nécessite plus que d'ajouter un seul tuple. Il évite donc les clés étrangères, toujours plus délicates à gérer d'un point de vue applicatif.

Remarque : Inconvénient du choix
Puisqu'il est rare que les thèses soient publiées sous forme de livre alors les tuples de la relation Document contiendront la plupart du temps une valeur nulle soit pour l'éditeur soit pour la discipline.
Cette introduction de valeurs nulles est moins problématique que l'introduction de redondance observée dans le cas précédent, aussi elle peut-être plus facilement tolérée. On considère alors que l'héritage est "presque" complet, puisque seuls deux attributs sont distingués. Cela dit ils s'agit de deux attributs sur quatre, soit une proportion importante d'une part, et il est regrettable que l'héritage soit également "presque" exclusif puisque l'introduction d'une valeur nulle sera quasi-systématique.

En conclusion

Conseil
L'héritage est toujours délicat à traduire en relationnel, ce qui est dommage car son pouvoir de représentation conceptuel est fort.
Un conseil pour assumer une gestion correcte de la traduction de la relation d'héritage serait d'appliquer à la lettre les règles de transformation (ce qui conduira le plus souvent à un héritage par référence) et, l'expérience aidant, de tolérer dans des cas bien maîtrisés des digressions à cette règle.

8. Comparaison des modèles E-A, UML et relationnel

Modèle E-A	Modèle UML	Modèle relationnel
Entité	Classe	Relation
Occurence d'une entité	Objet	Nuplet
Attribut simple	Attribut simple	Attribut atomique
Attribut composite	Attribut composite	Ensemble d'attributs
Attribut multivalué	Attribut multivalué	Relation
Attribut dérivé	Attribut dérivé	Procédure stockée ou contrainte dynamique
-	Méthode	Procédure stockée
Entité faible	-	Relation
Association 1:N	Association 1:N	Attributs
Association N:M	Association N:M	Relation
Association de degré 3 ou supérieur	Association de degré 3 ou supérieur	Relation
Occurence d'une association	Occurence d'une association	Nuplet
Héritage	Héritage	Vue

▲ TAB. 9 : SYNTHÈSE E-A ET UML VERS RELATIONNEL

Section A3. Le passage UML vers Relationnel

Afin de pouvoir implémenter une base de données, il faut pouvoir traduire le modèle conceptuel en modèle logique. Cela signifie qu'il faut pouvoir convertir un modèle UML en modèle relationnel. Les modèles conceptuels sont suffisamment formels pour ce passage soit systématisé.

1. Transformation des classes

Classe

Pour chaque classe C non abstraite, on crée une relation R dont le schéma est celui de la classe. La clé primaire de R est une des clés de C.

Les classes abstraites sont ignorées à ce stade, et n'étant pas instanciables, ne donnent jamais lieu à la création de relation.

2. Transformation des associations

Association 1:N

Pour chaque association binaire A de type 1:N (le cas échéant 0,1:N) entre les classes S et T (représentés par les relations RS et RT respectivement) on inclut dans la définition de RT comme clé étrangère la clé de RS.

Association M:N et associations de degré supérieur à 2

Pour chaque association binaire A de type M:N ou pour chaque association A de degré supérieur à 2, on crée une nouvelle relation RA pour représenter A. On met dans RA comme clé étrangère, les clés de toutes les relations correspondant aux classes participant à A et dont la concaténation formera sa clé.

Association 1:1

Pour chaque association binaire A de type 1,1:1,1 ou 1,1:1,0 entre les classes S et T (représentés par les relations RS et RT respectivement) on substitut dans la définition de RS l'éventuelle clé primaire (si RS était identifiée) par la clé primaire de RT qui est également définie comme clé étrangère vers RT. Notons que dans le cas d'une association 1,1:1,1 RT peut-être choisi à la place de RS pour accueillir la clé étrangère.

Pour chaque association binaire A de type 0,1:0,1 entre les classes S et T (représentés par les relations RS et RT respectivement) on inclut dans la définition de RS comme clé étrangère la clé de RT. La clé étrangère vers RT incluse dans RS est également définie comme étant unique, afin d'assurer la cardinalité maximum de 1. Notons qu'il n'était pas possible dans ce cas 0,1:0,1 que la clé étrangère vers

RT incluse dans RS soit également la clé primaire, car la cardinalité de 0 supposerait que la clé primaire puisse être "null", ce qui est illégal pour une clé.

Cardinalité minimale 0 ou 1
Selon que la cardinalité minimale est 0 ou 1 (ou plus) du côté de la relation référençante on ajoutera ou non une contrainte de non nullité sur la clé étrangère.
Selon que la cardinalité minimale est 0 ou 1 (ou plus) du côté de la relation référencée on ajoutera ou non une contrainte d'existance de tuples référençant pour chaque tuple de la relation référencée.

3. Remarques concernant la transformation des associations 1:1

Remarque : Choix de la relation référençante pour la traduction de l'association 1:1
La traduction d'une association 1:1 ou 0..1:0..1 conduit à devoir faire un choix entre les deux classes composant l'association, afin de déterminer laquelle est référencée de laquelle est référençante[Dans le cas d'une association 1:0..1, c'est toujours la classe coté 0..1 qui sera choisie pour référencer.]. Ce choix peut-être arbitraire, mais peut également être éclairé par la sémantique des relations, et l'on choisira alors celle qui est de plus faible importance du point de vue de la modélisation pour référencer (et donc intégrer la clé étrangère).

Exemple de choix de la relation référençante pour une relation 1:1
Soit deux classes, "homme" et "femme" et une association "mariage" de cardinalité 1:1 (hommes et femmes sont donc obligatoirement mariés) unissant ces deux classes. Les classes "homme" et "femme" sont identifiées par un attribut "nom". S'il serait plus galant de choisir "femme" comme classe principale, il sera dans la pratique plus opportun de choisir "homme", car le "nom" de la classe "homme" sera effectivement une propriété pertinente pour la classe "femme" et donc aura plus de sens en tant que clé primaire et en tant que clé étrangère (dans le cas inverse, la clé de l'homme serait le nom de sa femme, ce qui ne correspond pas à la réalité administrative du mariage). On choisira donc plutôt la représentation :

```
homme (nom)
femme (nomMariage=>homme, nomJeuneFille) avec nomJeuneFille clé
```

Si l'association avait été de cardinalité 0..1:0..1 (certains hommes et femmes ne sont pas mariés), le même choix se serait imposé et l'on aurait abouti au résultat suivant:

```
homme (nom)
femme (nomJeuneFille, nomMariage=>homme) avec nomMariage unique
```

Remarque : Fusion des relations dans le cas de la traduction de l'association 1:1
Il est possible, dans le cas d'une association 1:1 de décider de fusionner les deux relations en une seule. La relation résultante est alors composée de l'ensemble des attributs de chacune des deux relations, en choisissant l'une des deux clés de l'une des deux relations pour identifier la nouvelle relation[C'est également possible pour les associations 1:0..1, mais c'est généralement moins pertinent, puisque cela conduira à un certain nombre de champs null. Ce n'est jamais pertinent dans le cas de relations 0..1:0..1.].
Ce choix sera conduit par une appréciation du rapport entre :

- La complexité introduite par le fait d'avoir deux relations là ou une suffit,
- La pertinence de la séparation des deux relations d'un point de vue sémantique,
- Les pertes de performance dues à l'éclatement des relations,
- Les pertes de performance dues au fait d'avoir une grande relation,
- Les questions de sécurité et de sureté factorisées ou non au niveau des deux relations,
- etc.

Dans le doute il est préférable d'adopter l'approche systématique consistant à séparer les deux relations.

4. Transformation des attributs et méthodes

Attributs simples
Pour chaque attribut élémentaire et monovalué A d'une classe E (idem pour les associations), on crée un attribut correspondant à A sur la relation RE correspondant à E.

Attributs composites
Pour chaque attribut composite C, comprenant N sous-attributs, d'une classe E (idem pour les associations), on crée N attributs correspondants à C sur la relation RE correspondant à E.

Attributs multivalués
Pour chaque attribut multivalué M d'une classe E (idem pour les associations), on crée une nouvelle relation RM qui comprend un attribut monovalué correspondant à M plus la clé de RE (relation représentant E). La clé de RM est la concaténation des deux attributs.

Attributs dérivés et méthodes
On ne représente pas en général les attributs dérivés ni les méthodes dans le modèle relationnel, ils seront calculés dynamiquement soit par des procédures internes à la BD (procédures stockées), soit par des procédures au niveau applicatif.
On peut néanmoins décider (pour des raisons de performance essentiellement) de représenter l'attribut dérivé ou la méthode comme s'il s'agissait d'un attribut simple, mais il sera nécessaire dans ce cas d'ajouter des mécanismes de validation de contraintes dynamiques (avec des triggers par exemple) pour assurer que la valeur stockée évolue en même temps que les attributs sur lesquels le calcul dérivé porte. Notons qu'introduire un attribut dérivé ou un résultat de méthode dans le modèle relationnel équivaut à introduire de la redondance, ce qui est en gérénal déconseillé, et ce qui doit être dans tous les cas contrôlé.

5. Transformation des classes d'association

Classe d'association 1:N
Les attributs de la classe d'association sont ajoutés à la relation issue de la classe côté N.

Classe d'association N:M
Les attributs de la classe d'association sont ajoutés à la relation issue de l'association N:M.

Si la classe d'association possède une clé, celle-ci est concaténée aux clés étrangères composant déjà la clé primaire de la relation d'association.

Classe d'association 1:1
Les attributs de la classe d'association sont ajoutés à la relation qui a été choisie pour recevoir la clé étrangère. Bien entendu si les deux classes ont été fusionnées en une seule relation, les attributs sont ajoutée à celle-ci.

6. Transformation des compositions

Remarque :
Une composition est transformée comme une association 1:N, mais on ajoute à la clé de la classe partie (dite clé locale) la clé étrangère vers la classe composite.

Compositions
Soit la composition entre la classe composite C et la classe partie P (représentés par les relations RC et RP respectivement) on inclut dans la définition de RP comme clé étrangère la clé de RC. La clé de RP est redéfinie comme la concaténation de la clé de P (clé locale) avec la clé étrangère vers RC.

Remarque : Composition et entités faibles en E-A
Une composition est transformée selon les mêmes principes qu'une entité faible en E-A.

7. Transformation de la relation d'héritage

Le modèle relationnel ne permet pas de représenter directement une relation d'héritage, puisque que seuls les concepts de relation et de référence existent dans le modèle. Il faut donc appauvrir le modèle conceptuel pour qu'il puisse être représenté selon un schéma relationnel.

Trois solutions existent pour transformer une relation d'héritage :

- Représenter l'héritage par une référence entre la classe mère et la classe fille.
- Représenter uniquement les classes filles par des relations.
- Représenter uniquement la classe mère par une relation.

Choisir le bon mode de transformation d'une relation d'héritage

La difficulté consiste donc pour chaque relation d'héritage à choisir le bon mode de transformation, sachant que chaque solution possède ses avantages et ses inconvénients.

Mode de transformation	Limite	Cas d'usage	Exemple	Vue
Par référence	Lourdeur liée à la nécessité de représenter les données des classes filles sur deux relations	Adapté à tous les cas d'héritage ni exclusif, ni complet - particulièrement adapté lorsque la classe mère n'est pas abstraite.	Etudiant et Employe héritent de Personne (un étudiant peut être employé et les propriétés des étudiants et des employés sont différentes, de plus des personnes peuvent exister qui ne sont ni employés ni étudiants).	Une vue doit être créée pour chaque classe fille (réalisant la jointure avec la classe mère).
Par les classes filles	Redondance liée à l'existance simultanée de tuples dans plusieurs classes filles	Adapté à l'héritage exclusif - particulièrement adapté lorsque la classe mère est abstraite.	Homme et Femme héritent de Personne (un tuple de Homme ne peut pas être un tuple de Femme et Personne est abstraite, il n'existe pas de Personne qui ne soit ni un Homme, ni une Femme).	Une vue doit être créée, uniquement dans le cas où la classe mère n'est pas abstraite, pour unir les tuples des classes filles avec ceux spécifiques à la classe mère.
Par la classe mère	Nullité systématique pour les attributs d'une classe fille n'existant pas pour une autre classe fille (et pour la classe mère si celle-ci n'est pas abstraite)	Adapté à l'héritage complet - particulièrement adapté lorsque la classe mère n'est pas abstraite.	Responsable et Salarié héritent de Employé (un responsable est salarié et aucun attribut n'est spécifique ni à Responsable, ni à Salarié, de plus il existe des stagiaires par exemple qui sont juste employés, mais non salariés et non responsables)	Une vue doit être systématiquement créée pour chaque classe fille (réalisant la restriction et la projection depuis la relation unique créée).

▲ TAB. 10 : CHOIX DE LA BONNE TRANSFORMATION

Mode de transformati	Classe mère abstraite	Classe mère non abstraite	Héritage exclusif	Héritage non exclusif	Héritage complet	Héritage presque complet	Héritage non complet
Par référence	- -	+	=	=	-	=	=
Par les classes filles	+	-	+ +	- -	=	=	=
Par la classe mère	-	+	+.	=	+ +	=	-

▲ TAB. 11 : CHOIX DE LA BONNE TRANSFORMATION

8. Transformation de la relation d'héritage par référence

Héritage représenté par une référence

Chaque classe, mère ou fille, est représentée par une relation. La clé primaire d'une classe mère est utilisée pour identifier chacune de ses classes filles (cette clé étant pour chaque classe fille à la fois la clé primaire et une clé étrangère vers la classe générale).
Si une classe fille avait une autre clé primaire candidate dans le modèle conceptuel, cette clé n'est pas retenue pour être la clé primaire, étant donné que c'est la clé étrangère référence à la classe mère qui est retenue.
La cardinalité d'un lien entre une classe fille et une classe mère est (1,1):(0,1). En effet toute classe fille référence obligatoirement une et une seule classe mère (pas d'héritage multiple) et toute classe mère est référencée une ou zéro fois (zéro fois si un objet peut être directement du type de la classe mère) par chaque classe fille .
Une vue devra être créée pour chaque classe fille en réalisant une jointure sur la classe mère.

Exemple : Héritage représenté par une référence

Soit la classe A avec la clé K et les attributs A1 et A2. Soit la classe B, classe fille de A, comprenant la clé K' et les attributs B1 et B2. Le modèle relationnel correspondant selon cette transformation est [on remarquera que K' n'est pas retenue comme clé primaire de B] :

```
A (K, A1, A2)
B (K=>A, K', B1, B2)
```

Remarque : Classe mère non abstraite

Cette solution est particulièrement adaptée lorsque la classe mère n'est pas abstraite, car cela en autorise l'instanciation sans aucun bruit dans la relation lié aux classes filles. Par contre si la classe mère est abstraite il faut introduire une contrainte dynamique complexe pour imposer la présence d'un tuple référençant dans une des classes filles.
Ainsi dans l'exemple précédent, un A peut être instancié en toute indépendance par rapport à la relation B.

Solution générale

Cette solution est la plus générale, elle fonctionne correctement dans tous les cas. Le seul problème est la complexité de la structure de donnée qu'elle induit.

9. Transformation de la relation d'héritage par les classes filles

Héritage absorbé par les classes filles

Chaque classe fille est représenté par une relation. Tous les attributs de la classe mère sont répétés au niveau de chaque classe fille .
Si une classe fille a une clé primaire propre, cette clé n'est pas retenue, et c'est bien la clé héritée de la classe mère qui devient la clé primaire (mais la clé est bien entendu maintenue comme clé candidate)
Si la classe mère n'est pas abstraite, elle est également représentée par une relation qui ne contiendra que les tuples qui ne sont pas aussi des instances de classes filles. La classe mère est alors représentée par une vue qui unit ses propres tuples avec les tuples de toutes ses classes filles et en les projetant sur les attributs hérités uniquement.

Exemple : Héritage absorbé par les classes filles

Soit la classe abstraite A avec la clé K et les attributs A1 et A2. Soit la classe B, classe fille de A avec les attributs B1 et B2. Soit la classe C, classe fille de A avec les attributs C1 et C2 Le modèle relationnel correspondant selon cette transformation est :

```
B (K, A1, A2, B1, B2)
C (K, A1, A2, C1, C2)
```

Si A n'avait pas été abstraite elle aurait donné naissance à une relation A possédant les attributs A1 et A2 et qui n'auraient alors contenu que les tuples qui ne sont ni des B ni des C.

Remarque : Classe mère abstraite
Cette solution est particulièrement adaptée lorsque la classe mère est abstraite, car elle permet de ne pas matérialiser cette classe mère par une relation.
En revanche dans le cas où la classe mère n'est pas abstraite, la solution est moins adaptée, car il faut alors passer par une vue pour retrouver tous les tuples de la classe.

Héritage exclusif
Cette solution est optimum dans le cas d'un héritage exclusif, c'est à dire si aucun objet d'une classe fille n'appartient aussi à une autre classe fille. Dans le cas contraire, le problème est que des redondances vont être introduites puisqu'un même tuple devra être répété pour chaque classe fille à laquelle il appartient.

10. Transformation de la relation d'héritage par la classe mère

Héritage absorbé par la classe mère
La classe mère est représentée par une relation, mais ses classes filles ne sont pas représentées par des relations. Tous les attributs de chaque classe fille sont réintégrés au niveau de la classe mère.
Un attribut supplémentaire, dit de discrimination, est ajouté à la classe mère, afin de distinguer les tuples des différentes classes filles. Cet attribut est de type énumération et a pour valeurs possibles les noms des différents classes filles. Afin de gérer l'héritage non exclusif (un objet peut être de plusieurs classe fille à la fois), le domaine de l'attribut de discrimination, peut être étendu à des combinaisons de noms des différents classes filles. Si la classe mère n'est pas abstraite, une valeur supplémentaire peut-être ajoutée au domaine, ou bien l'on peut autoriser la valeur null qui désignera alors un objet de la classe mère.
Si une classe fille a une clé primaire propre, cette clé sera réintégrée à la classe mère, au même titre qu'un autre attribut, mais bien entendu n'officiera pas en tant que clé primaire (et en général pas non plus en tant que clé candidate car elle pourra contenir des valeurs nulles).
Chaque classe fille est représentée par une vue qui sélectionne les tuples de la classe mère qui appartiennent à la classe fille et les projete sur les attributs de la classe fille.

Exemple : Héritage absorbé par la classe mère
Soit la classe A avec la clé K et les attributs A1 et A2. Soit la classe B, classe fille de A avec les attributs B1 et B2. Soit la classe C, classe fille de A avec les attributs C1 et C2 Le modèle relationnel correspondant selon cette transformation est [on remarquera l'attibut supplémentaire de discrimination D dont le domaine est l'énumération des noms des classes filles, avec ici la prise en compte d'un héritage non exclusif.] :

```
A (K, A1, A2, B1, B2, C1, C2, D:{'B','C','BC'})
```

Si l'on pose que A n'est pas abstraite, alors un tuple sera un A s'il a la valeur null pour sa propriété D. Si l'on pose que A est abstraite, on ajoutera la contrainte NOT NULL à la propriété D.

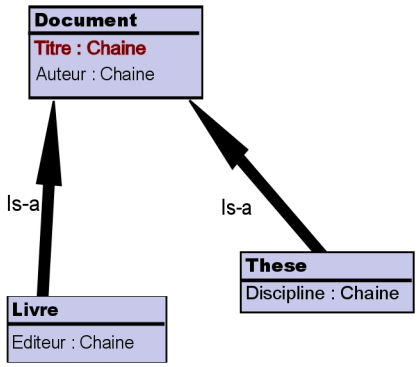
Remarque : Classe mère non abstraite
Cette solution est particulièrement adaptée lorsque la classe mère n'est pas abstraite, car cela en autorise l'instanciation naturellement, en ne renseignant pas l'attribut de discrimination. Lorsque la classe mère est abstraite des contraintes supplémentaires doivent être ajoutées pour assurer que l'appartenance à une classe fille est toujours spécifiée.

Héritage complet
Cette solution est optimum dans le cas d'un héritage complet, c'est à dire si les classes filles ne définissent pas d'attributs autre que ceux hérités. Dans le cas contraire cela engendre des valeurs null.
En pratique l'héritage complet est rare, mais nombre d'héritages le sont presque et pourront alors être raisonnablement gérés sellon cette approche, a fortiori si la classe mère n'est pas abstraite.

Autre représentation de la discrimination
Si l'héritage concerne un nombre élevé de classes filles et qu'il est principalement non exclusif alors le domaine de l'attribut de discrimination peut impliquer une combinatoire importante de valeurs. Pour contourner cette lourdeur il est possible d'utiliser, en remplacement, un attribut de domaine booléen pour chaque classe fille spécifiant si un tuple est un objet de cette classe.
Dans cette configuration la classe mère sera logiquement considérée comme non abstraite et un objet appartiendra à la classe mère si tous ses attributs de discrimination valent "faux". Seule une contrainte dynamique permettra de définir la classe mère comme abstraite, en précisant que les attributs de discrimination ne peuvent être tous à "faux".

11. Exemple de transformation d'une relation d'héritage

Exemple
Soit le modèle UML suivant :



▲ SCH. 6 : REPRÉSENTATION DE DOCUMENTS

Il existe trois façons de traduire la relation d'héritage : par référence, par absorption par les classes filles, par absorption par la classe mère.

Remarque : Type d'héritage
Si une thèse peut également être un livre (et dans la réalité c'est bien le cas puisqu'une thèse peut-être publiée), alors l'héritage n'est pas exclusif puisque un même document peut être une thèse *et* un livre.
L'héritage n'est pas non plus complet, puisque l'on observe que les thèses et les livres ont des attributs qui ne sont pas communs (la discipline pour la thèse et l'éditeur pour le livre).
En toute rigueur, il faudrait donc appliquer le cas général (ni exclusif, ni complet) et appliqué une traduction de l'héritage par référence. Observons néanmoins les avantages et inconvénients que chacun des trois choix porterait.

Héritage représenté par une référence

```
Document(Titre:Chaîne, Auteur:Chaîne)
These(Titre=>Document, Discipline:Chaîne)
Livre(Titre=>Document), Editeur:Chaîne
```

Remarque : Avantages du choix
Il n'y a ni redondance ni valeur nulle inutile dans les relations These et Livre, c'est la traduction la plus juste pour le problème posé.

Remarque : Inconvénient du choix
Il est relativement lourd de devoir créer un tuple dans Document pour chaque tuple dans These ou Livre, alors que la relation Document ce porte que l'information concernant l'auteur, juste pour assurer les cas, par ailleurs plutôt rare dans la réalité, où les thèses sont publiées sous forme de livres.

Héritage absorbé par les classes filles

```
These(Titre, Discipline:Chaîne, Auteur:Chaîne)
Livre(Titre), Editeur:Chaîne, Auteur:Chaîne
```

Remarque : Avantages du choix
Il est plus simple que le précédent, puisque la représentation d'une thèse ou d'un livre ne nécessite plus que d'ajouter un seul tuple. Il évite donc les clés étrangères, toujours plus délicates à gérer d'un point de vue applicatif.

Remarque : Inconvénient du choix
Lorsqu'une thèse est publiée sous la forme d'un livre, alors une information sera dupliquée (l'auteur), ce qui introduit de la redondance. Si une telle redondance peut-être tolérée pour des raisons de simplification ou de performance (si on considère que le cas est rare par exemple et donc que l'héritage est "presque" exclusif), il sera néanmoins nécessaire d'assurer son contrôle par un moyen supplémentaire.
Dans le cas général, il n'est pas souhaitable de tolérer une telle redondance.

Héritage absorbé par la classe mère

```
Document(Titre, Discipline:Chaîne, Editeur:Chaîne, Auteur:Chaîne,
Type:{These|Livre|These+Livre})
```

Remarque : Avantages du choix
Il est plus simple que le premier, puisque la représentation d'une thèse ou d'un livre ne nécessite plus que d'ajouter un seul tuple. Il évite donc les clés étrangères, toujours plus délicates à gérer d'un point de vue applicatif.

Remarque : Inconvénient du choix
Puisqu'il est rare que les thèses soient publiées sous forme de livre alors les tuples de la relation Document contiendront la plupart du temps une valeur nulle soit pour l'éditeur soit pour la discipline.
Cette introduction de valeurs nulles est moins problématique que l'introduction de redondance observée dans le cas précédent, aussi elle peut-être plus facilement tolérée. On considère alors que l'héritage est "presque" complet, puisque seuls deux attributs sont distingués. Cela dit ils s'agit de deux attributs sur quatre, soit une proportion importante d'une part, et il est regrettable que l'héritage soit également "presque" exclusif puisque l'introduction d'une valeur nulle sera quasi-systématique.

En conclusion

Conseil
L'héritage est toujours délicat à traduire en relationnel, ce qui est dommage car son pouvoir de représentation conceptuel est fort.
Un conseil pour assumer une gestion correcte de la traduction de la relation d'héritage serait d'appliquer à la lettre les règles de transformation (ce qui conduira le plus souvent à un héritage par référence) et, l'expérience aidant, de tolérer dans des cas bien maîtrisés des digressions à cette règle.

12. Héritage et clé primaire

Dans tous les cas, notamment en cas d'héritage à plusieurs niveaux, c'est la clé primaire de la classe la plus générale qui est retenue pour identifier les classes filles héritant directement ou indirectement de cette classe.

Exemple :
Ainsi si C hérite de B qui hérite de A, c'est la clé de A qui permettra d'identifier les classes A, B et C, et ce quelque soit le mode de transformation retenu.

13. Liste des contraintes

Contraintes exprimées sur le diagramme
Les contraintes exprimées sur le diagrammes sont reportées dans un tableau qui accompagnera le modèle logique.

Extension des contraintes exprimées
On s'attachera lors de la modélisation logique à exprimer l'ensemble des contraintes dynamiques pesant sur le modèle, même celles qui ont été considérées comme secondaires ou évidentes lors de la modélisation conceptuelle.

14. Correspondance entre UML et relationnel

Modèle UML	Modèle relationnel
Classe instanciable	Relation
Classe abstraite	Rien
Objet	Nuplet
Attribut simple	Attribut atomique
Attribut composite	Ensemble d'attributs
Attribut multivalué	Relation
Attribut dérivé	Procédure stockée ou contrainte dynamique
Méthode	Procédure stockée
Association 1:N	Attributs
Association N:M	Relation
Association de degré 3 ou supérieur	Relation
Composition	Attributs
Classe d'association	Attributs
Occurence d'une association	Nuplet
Héritage	Vues

▲ TAB. 12 : SYNTHÈSE UML VERS RELATIONNEL

Chapitre B. Algèbre relationnel

Objectifs pédagogiques

Connaître les fondements du modèle relationnel.
Savoir effectuer le passage d'un schéma conceptuel à un schéma relationnel.
Maîtriser l'algèbre relationnelle.

Section B1. Algèbre relationnelle

Concepts manipulatoires

La représentation d'information sous forme relationnelle est intéressante car les fondements mathématiques du relationnel, outre qu'ils permettent une modélisation logique simple et puissante, fournissent également un ensemble de concepts pour manipuler formellement l'information ainsi modélisée.

Ainsi une algèbre relationnelle, sous forme d'un ensemble d'opérations formelles permet d'exprimer des questions, ou requêtes, posées à une représentation relationnelle, sous forme d'expressions algébriques.
L'algèbre relationnelle est composée par les cinq opérateurs de base et les trois opérateurs additionnels suivants :

- **Opérateurs de base**
  - Union
  - Différence
  - Projection
  - Restriction
  - Produit cartésien

- **Opérateurs additionels**
  - Intersection
  - Jointure
  - Division

Remarque : Algèbre relationnelle et SQL
Les questions formulées en algèbre relationnelle sont la base du langage SQL, qui est un langage informatique d'expressions permettant d'interroger une base de données relationnelle.

Remarque : Algèbre relationnelle et objet
L'algèbre relationnelle est étendue à une algèbre permettant de manipuler des objets autres, et a priori plus complexes, que des relations. Cette algèbre étendue permet l'interrogation d'informations modélisées sous forme relationnelle-objet.

1. Opérateurs ensemblistes

Attention
Les opérateurs ensemblistes sont des relations binaires (c'est à dire entre deux relations) portant sur des relations *de même schéma*.

Union
L'union de deux relations R1 et R2 de même schéma produit une relation R3 de même schéma constituée de l'ensemble des tuples appartenant à R1 et/ou à R2.

Différence
La différence entre deux relations R1 et R2 de même schéma produit une relation R3 de même schéma constituée de l'ensemble des tuples de R1 n'appartenant pas à R2. Notons que la différence entre R1 et R2 n'est pas égale à la différence entre R2 et R1.

Intersection
L'intersection de deux relations R1 et R2 de même schéma produit une relation R3 de même schéma constituée de l'ensemble des tuples appartenant à la fois à R1 et à R2. Notons que l'intersection n'est pas une opération de base, car elle est équivalent à deux opérations de différence successives.

Exemple
Soit les deux relations suivantes :

```
Homme (Nom, Prénom, Age)
Femme (Nom, Prénom, Age)
```

Soit les tuples suivants pour ces deux relations respectivement :

```
(Dupont, Pierre, 20)
(Durand, Jean, 30)

(Martin, Isabelle, 20)
(Tintin, Hélène, 30)
```

Soit l'opération suivante :

```
R = Union (Homme, Femme)
```

On obtient alors la relation R composée des tuples suivants :

```
(Dupont, Pierre, 20)
(Durand, Jean, 30)
(Martin, Isabelle, 20)
(Tintin, Hélène, 30)
```

La différence entre Homme et Femme (respectivement entre Femme et Homme) renvoie la relation Homme (respectivement Femme), car aucun tuple n'est commun aux deux relations. L'intersection entre Homme est Femme est vide, pour la même raison.

Remarque : Union externe
Il est possible de définir une opération d'union externe, qui permet de réaliser l'union de deux relations de schéma différent, en ramenant les relations aux mêmes schémas, et en les complétant avec des valeurs nulles.

2. Projection

Projection
La projection est une opération unaire (c'est à dire portant sur une seule relation). La projection de R1 sur une partie de ses attributs {A1, A2, ...} produit une relation R2 dont le schéma est restreint aux attributs mentionnés en opérande, comportant les mêmes tuples que R1, et dont les doublons sont éliminés.

Remarque : Elimination des doublons
Après suppression d'une partie des attributs du schéma, la relation peut comporter des doublons. Etant donné que l'on ne pourrait plus identifier ces doublons les uns par rapport aux autres, la seule solution sensée est donc de considérer que deux doublons sont équivalents, et donc de n'en garder qu'un seul dans la relation résultante.

Exemple
Soit la relation suivante :

```
Personne (Nom, Prénom, Age)
```

Soit les tuples suivants :

```
(Dupont, Pierre, 20)
(Durand, Jean, 30)
```

Soit l'opération suivante :

```
R = Projection (Personne, Nom, Age)
```

On obtient alors la relation R composée des tuples suivants :

```
(Dupont, 20)
(Durand, 30)
```

3. Restriction

Restriction
La restriction est une opération unaire (c'est à dire portant sur une seule relation). La restriction de R1, étant donnée une condition C, produit une relation R2 de même schéma que R1 et dont les tuples sont les tuples de R1 vérifiant la condition C.

Exemple
Soit la relation suivante :

```
Personne (Nom, Prénom, Age)
```

Soit les tuples suivants :

```
(Dupont, Pierre, 20)
(Durand, Jean, 30)
```

Soit l'opération suivante :

```
R = Restriction (Personne, Age>25)
```

On obtient alors la relation R composée de l'unique tuple restant suivant :

```
(Durand, Jean, 30)
```

4. Produit

Produit cartésien

Le produit cartésien est une opération binaire (c'est à dire portant sur deux relations). Le produit de R1 par R2 (équivalent au produit de R2 par R1) produit une relation R3 ayant pour schéma la juxtaposition de ceux des relations R1 et R2 et pour tuples l'ensemble des combinaisons possibles entre les tuples de R1 et ceux de R2.

- *Synonyme : Produit.*

Remarque

Le nombre de tuples résultant du produit de R1 par R2 est égal au nombre de tuples de R1 multiplié par le nombre de tuples de R2.

Exemple

Soit les deux relations suivantes :

```
Homme (Nom, Prénom, Age)
Femme (Nom, Prénom, Age)
```

Soit les tuples suivants pour ces deux relations respectivement :

```
(Dupont, Pierre, 20)
(Durand, Jean, 30)

(Martin, Isabelle, 15)
(Tintin, Hélène, 40)
```

Soit l'opération suivante :

```
R = Produit (Homme, Femme)
```

On obtient alors la relation R composée des tuples suivants :

```
(Dupont, Pierre, 20, Martin, Isabelle, 15)
(Durand, Jean, 30, Martin, Isabelle, 15)
(Dupont, Pierre, 20, Tintin, Hélène, 40)
(Durand, Jean, 30, Tintin, Hélène, 40)
```

5. Jointure

Jointure

La jointure est une opération binaire (c'est à dire portant sur deux relations). La jointure de R1 et R2, étant donné une condition C portant sur des attributs de R1 et de R2, *de même domaine*, produit une relation R3 ayant pour schéma la juxtaposition de ceux des relations R1 et R2 et pour tuples l'ensemble de ceux obtenus par concaténation des tuples de R1 et de R2, et qui vérifient la condition C.

Exemple

Soit les deux relations suivantes :

```
Homme (Nom, Prénom, Age)
Enfant (Nom, Prénom, Age)
```

Soit les tuples suivants pour ces deux relations respectivement :

```
(Dupont, Pierre, 20)
(Durand, Jean, 30)

(Dupont, Georges, 1)
(Dupont, Jacques, 3)
```

Soit l'opération suivante :

```
R = Jointure (Homme, Enfant, Homme.Nom=Enfant.Nom)
```

On obtient alors la relation R composée des tuples suivants :

```
(Dupont, Pierre, 20, Dupont, Georges, 1)
(Dupont, Pierre, 20, Dupont, Jacques, 3)
```

Remarque : Opération additionnelle
La jointure n'est pas une opération de base, elle peut être réécrite en combinant le produit et la restriction.

6. Jointure naturelle

Jointure naturelle
La jointure naturelle entre R1 et R2 est une jointure pour laquelle la condition est l'égalité entre les attributs de même nom de R1 et de R2. Il est donc inutile de spécifier la condition dans une jointure naturelle, elle reste toujours implicite.

Exemple
Soit deux relations R1 (A, B, C) et R2 (A, D), l'opération Jointure(R1,R2,R1.A=R2.A) est équivalente à l'opération JointureNaturelle(R1,R2).

Remarque
Pour appliquer une jointure naturelle, il faut que les deux relations opérandes aient au moins un attribut ayant le même nom en commun.

7. Jointure externe

La jointure est une opération qui entraîne la perte de certains tuples : ceux qui appartiennent à une des deux relations opérandes et qui n'ont pas de correspondance dans l'autre relation. Il est nécessaire dans certains cas de palier cette lacune, et l'on introduit pour cela la notion de jointure externe.

Jointure externe
La jointure externe entre R1 et R2 est une jointure qui produit une relation R3 à laquelle on ajoute les tuples de R1 et de R2 exclus par la jointure, en complétant avec des valeurs nulles pour les attributs de l'autre relation.

Jointure externe gauche
La jointure externe gauche entre R1 et R2 est une jointure externe pour laquelle on ajoute seulement les tuples de R1 (c'est à dire la relation de gauche) ayant été exclus.

- *Synonyme : Jointure gauche.*

Jointure externe droite
La jointure externe droite entre R1 et R2 est une jointure externe pour laquelle on ajoute seulement les tuples de R2 (c'est à dire la relation de droite) ayant été exclus.

Bien entendu une jointure externe droite peut être réécrite par une jointure externe gauche (et réciproquement) en substituant les relations opérandes R1 et R2.

- *Synonyme : Jointure droite.*

Exemple
Soit les deux relations suivantes :

```
Homme (Nom, Prénom, Age)
Enfant (Nom, Prénom, Age)
```

Soit les tuples suivants pour ces deux relations respectivement :

```
(Dupont, Pierre, 20)
(Durand, Jean, 30)


(Dupont, Georges, 1)
(Martin, Isabelle, 15)
```

Soit l'opération suivante :

```
R = JointureExterne (Homme, Enfant, Homme.Nom=Enfant.Nom)
```

On obtient alors la relation R composée des tuples suivants :

```
(Dupont, Pierre, 20, Dupont, Georges, 1)
(Durand, Jean, 30, Null, Null, Null)
(Null, Null, Null, Martin, Isabelle, 15)
```

Une jointure externe gauche n'aurait renvoyé que les deux premiers tuples et une jointure externe droite n'aurait renvoyée que le premier et le troisième tuple.

8. Division

Division

La division est une opération binaire (c'est à dire portant sur deux relations). La division de R1 par R2, sachant que R1 et R2 ont au moins un attribut commun (c'est à dire de même nom et de même domaine), produit une relation R3 qui comporte les attributs appartenant à R1 mais n'appartenant pas à R2 et l'ensemble des tuples qui concaténés à ceux de R2 donnent toujours un tuple de R1.

Exemple

Soit les deux relations suivantes :

```
Homme (Nom, Prénom, Métier)
Métier (Metier)
```

Soit les tuples suivants pour ces deux relations respectivement :

```
(Dupont, Pierre, Ingénieur)
(Dupont, Pierre, Professeur)
(Durand, Jean, Ingénieur)


(Ingénieur)
(Professeur)
```

Soit l'opération suivante :

```
R = Division (Homme, Métier)
```

On obtient alors la relation R composée des tuples suivants :

```
(Dupont, Pierre)
```

Remarque : Réponse aux questions : Pour tous les ...

La division permet de répondre aux questions du type : "Donnez toutes les personnes qui pratiquent tous les métiers de la relation métier".

Remarque : Opération additionnelle

La division n'est pas une opération de base, elle peut être réécrite en combinant le produit, la restriction et la différence.

9. Proposition de notations

Il existe plusieurs syntaxes pour écrire des opérations d'algèbre relationnelle, certaines inspirées de l'algèbre classiques, d'autres reposant sur des notations graphiques. Nous proposons une notation fonctionnelle qui a le mérite d'être facile à écrire et d'être lisible. Si cette notation peut parfois perdre en simplicité, lorsqu'elle concerne un nombre élevé d'opérateurs, il est possible de décomposer une opération compliquée afin de l'alléger.

Syntaxe

```
R = Union (R1, R2)
R = Différence (R1, R2)
R = Intersection (R1, R2)
R = Projection (R1, A1, A2, ...)
R = Restriction (R1, condition)
R = Produit (R1, R2)
R = Jointure (R1, R2, condition)
R = JointureNaturelle (R1, R2)
R = JointureExterne (R1, R2, condition)
R = JointureGauche (R1, R2, condition)
R = JointureDroite (R1, R2, condition)
R = Division (R1, R2)
```

Exemple : Notation synthétique

```
R = Projection( Restriction(R1, A1=1 AND A2=2), A3)
```

Exemple : Notation décomposée

```
R' = Restriction(R1, A1=1 AND A2=2)
R = Projection (R', A3)
```

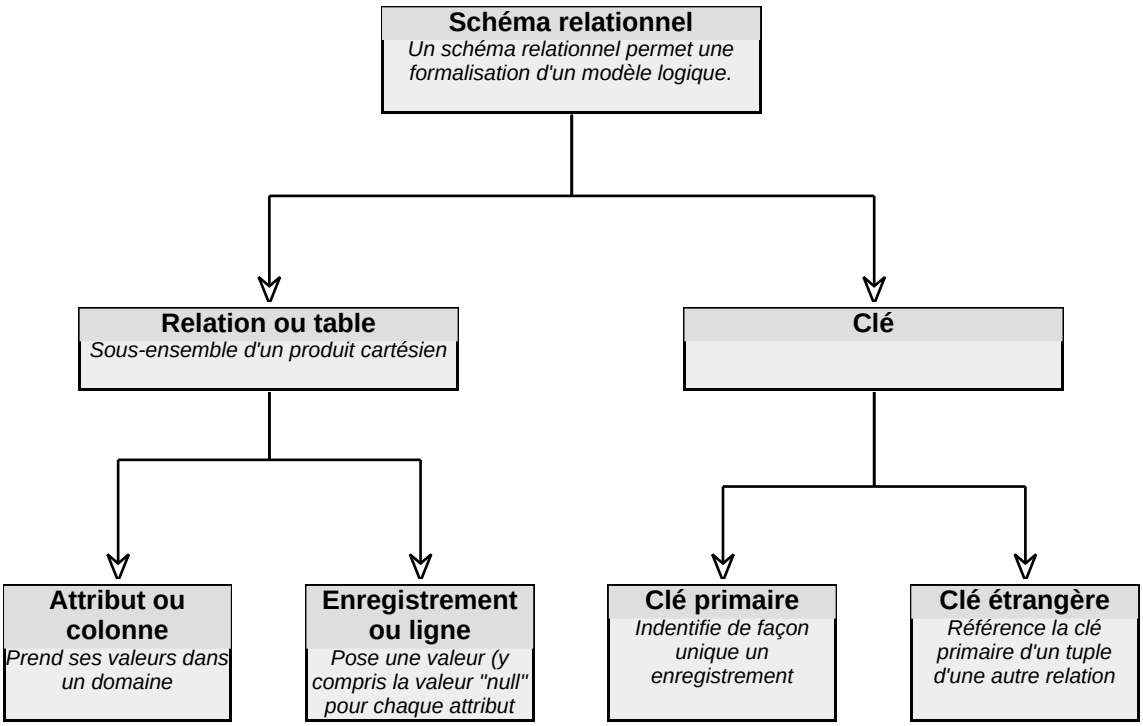
Chapitre C. Questions/réponses sur le relationnel

Question/Réponse 1. Transformation des associations 1:1

Dans le passage conceptuel vers relationnel, pour la transformation de l'association 1:1 entre 2 entités S et R, il est dit que l'on substitue dans la définition de RS l'éventuelle clé primaire par la clé primaire de RT qui est également clé étrangère vers RT. Doit-on néanmoins garder la clé de départ comme simple attribut ?

Il faut bien sûr garder la clé primaire originale comme attribut, et en tant que clé candidate non primaire (UNIQUE NOT NULL).

En résumé...



Pour aller plus loin ...

"http://eric.univ-lyon2.fr/~jdarmont/docs/sise-bd-exam0203.pdf", Examen de bases de données, **DARMONT J,** janvier, 2004.

Des exercices corrigés sur la modélisation UML et le relationnel.

"http://eric.univ-lyon2.fr/~jdarmont/docs/sise-bd-ex_rel.pdf", Exercices - Modèle Relationnel, **DARMONT J,** janvier, 2004.

Des exercices corrigés sur le relationnel.

DELMAL P, "SQL2 SQL3, applications à Oracle", De Boeck Université, 2001.

Une définition synthétique et efficace du domaine relationnel : relation, domaine, attribut, clé, intégrité, opérateurs (Premier chapitre).

"http://www.info.uqam.ca/~godin/DiaposParChap/ChapI-5.ppt", Introduction au modèle relationnel, **GODIN R,** avril, 2004.

Un rappel du modèle logique relationnel, et du passage d'un modèle conceptuel UML au relationnel.

Cours général

III

SQL LMD

SQL est un langage standardisé, implémenté par tous les SGBDR, qui permet, indépendemment de la plate-forme technologique et de façon déclarative, de définir le modèle de données, de le contrôler et enfin de le manipuler.
Ce chapitre traite du LMD de SQL, c'est à dire de la partie de ce langage, fondé sur l'algèbre relationnelle, qui permet d'interroger les données stockées en BD.

Chapitre A. Manipulation de données

Objectifs pédagogiques
Maîtriser les bases du SQL pour écrire des questions exigeant des jointures, projections, restriction, des tris et des agrégats.
Etre capable d'apprendre des notions particulières de SQL liés à un SGBD en particulier ou à des évolutions futures de SQL.

Qu'appelle-t-on SQL?

[Copyright 2003 Jean-François Pillou - Ce document issu de CommentCaMarche.net est soumis à la licence GNU FDL] SQL (pour langage de requêtes structuré) est un langage de définition de données (LDD, ou en anglais DDL Data Definition Language), un langage de manipulation de données (LMD, ou en anglais DML, Data Manipulation Language), et un langage de contrôle de données (LCD, ou en anglais DCL, Data Control Language), pour les bases de données relationnelles.

Le modèle relationnel a été inventé par E.F. Codd (Directeur de recherche du centre IBM de San José) en 1970, suite à quoi de nombreux langages ont fait leur apparition :

- IBM Sequel (Structured English Query Language) en 1977
- IBM Sequel/2
- IBM System/R
- IBM DB2

Ce sont ces langages qui ont donné naissance au standard SQL, normalisé en 1986 par l'ANSI pour donner SQL/86. Puis en 1989 la version SQL/89 a été approuvée. La norme SQL/92 a désormais pour nom SQL 2.

Remarque : SQL est un langage de définition de données
SQL est un langage de définition de données (LDD), c'est-à-dire qu'il permet de créer des tables dans une base de données relationnelle, ainsi que d'en modifier ou en supprimer.

Remarque : SQL est un langage de manipulation de données
SQL est un langage de manipulation de données (LMD), cela signifie qu'il permet de sélectionner, insérer, modifier ou supprimer des données dans une table d'une base de données relationnelle.

Remarque : SQL est un langage de protections d'accès
Il est possible avec SQL de définir des permissions au niveau des utilisateurs d'une base de données. On parle de DCL (Data Control Language).

Section A1. Le LMD de SQL

1. Sélection

La requête de sélection ou question est la base de la recherche de données en SQL.

Sélection
La selection est la composition d'un produit cartésien, d'une restriction et d'une projection (ou encore la composition d'une jointure et d'une projection).

Syntaxe

```
SELECT <liste d'attributs projetés>
FROM <liste de relations>
WHERE <condition>
```

La partie SELECT indique le sous-ensemble des attributs qui doivent apparaître dans la réponse (c'est le schéma de la relation résultat).
La partie FROM décrit les relations qui sont utilisables dans la requête (c'est à dire l'ensemble des attributs que l'on peut utiliser).
La partie WHERE exprime les conditions que doivent respecter les attributs d'un tuple pour pouvoir être dans la réponse. Une condition est un prédicat et par conséquent renvoie un booléen. Cette partie est optionnelle.
Afin de décrire un attribut d'une relation dans le cas d'une requête portant sur plusieurs relations, on utilise la notation "RELATION.ATTRIBUT".

Exemple

```
SELECT Nom, Prenom
FROM Personne
WHERE Age>18
```

Cette requête sélectionne les attributs Nom et Prenom des tuples de la relation Personne, ayant un attribut Age supérieur à 18.

Exemple

```
SELECT Parent.Prenom, Enfant.Prenom
FROM Parent, Enfant
WHERE Enfant.Nom=Parent.Nom
```

Cette requête sélectionne les prénoms des enfants et des parents ayant le même nom. On remarque la notation Parent.Nom et Enfant.Nom pour distinguer les attributs Prenom des relations Parent et Enfant.
On notera que cette sélection effectue une jointure sur les propriétés Nom des relations Parent et Enfant.

Remarque : SELECT *
Pour projeter l'ensemble des attributs d'une relation, on peut utiliser le caractère "*" à la place de la liste des attributs à projeter.

Exemple

```
SELECT *
FROM Avion
```

Cette requête sélectionne tous les attributs de la relation Avion.
Notons que dans cet exemple, la relation résultat est exactement la relation Avion

Remarque : SELECT DISTINCT
L'opérateur SELECT n'élimine pas les doublons (i.e. les tuples identiques dans la relation résultat) par défaut. Il faut pour cela utiliser l'opérateur "SELECT DISTINCT".

Exemple

```
SELECT DISTINCT Avion
FROM Vol
WHERE Date=31-12-2000
```

Cette requête sélectionne l'attribut Avion de la relation Vol, concernant uniquement les vols du 31 décembre 2000 et renvoie les tuples dans doublons.

Remarque : Renommage de propriété

Il est possible de redéfinir le nom des propriétés de la relation résultat. Ainsi "SELECT p AS p' FROM relation" renvoie une relation ayant comme propriété p'. Cette possibilité est offerte à partir de SQL2 (niveau entrée).

Remarque : Projection de constante

Il est possible de projeter directement des constantes. Ainsi "SELECT 'Bonjour' AS Essai" renverra un tuple avec une propriété Essai à la valeur 'Bonjour'.

2. Opérateurs de comparaisons et opérateurs logiques

La clause WHERE d'une instruction de sélection est définie par une condition. Une telle condition s'exprime à l'aide d'opérateurs de comparaison et d'opérateurs logiques. Le résultat d'une expression de condition est toujours un booléen.

Condition

```
Condition Elémentaire ::= Propriété <Opérateur de comparaison> Constante
Condition ::= Condition <Opérateur logique> Condition | Condition
Elémentaire
```

Les opérateurs de comparaison sont :

- P = C
- P <> C
- P < C
- P > C
- P <= C
- P >= C
- P BETWEEN C1 AND C2
- P LIKE 'chaîne'
- P IN (C1, C2, ...)
- P IS NULL

Les opérateur logique sont :

- OR
- AND
- NOT

3. Expression du produit cartésien

Syntaxe

```
SELECT *
FROM R1, R2, Ri
```

4. Expression d'une projection

Syntaxe

```
SELECT P1, P2, Pi
FROM R
```

5. Expression d'une restriction

Syntaxe

```
SELECT *
FROM R
WHERE <condition>
```

6. Expression d'une jointure

Syntaxe : Jointure par la clause WHERE

En tant que composition d'un produit cartésien et d'une restriction la jointure s'écrit alors :

```
SELECT *
FROM R1, R2, Ri
WHERE <condition>
```

Avec Condition permettant de joindre des attributs des Ri

Syntaxe : Jointure par la clause ON

On peut également utiliser la syntaxe dédiée suivante :

```
SELECT *
FROM R1 INNER JOIN R2
ON <condition>
```

Et pour plusieurs relations :

```
SELECT *
FROM (R1 INNER JOIN R2 ON <condition>) INNER JOIN Ri ON <condition>
```

Exemple : Une jointure naturelle

```
SELECT *
FROM R1, R2
WHERE R2.NUM = R1.NUM
```

Remarque : Auto-jointure

Pour réaliser une auto-jointure, c'est à dire la jointure d'une relation avec elle-même, on doit utiliser le renommage des relations. Pour renommer une relation, on note dans la clause FROM le nom de rennomage après le nom de la relation : "FROM NOM_ORIGINAL NOUVEAU_NOM".

Exemple : Auto-jointure

```
SELECT E1.Nom
FROM Employe E1, Employe E2
WHERE E1.Nom= E2.Nom
```

Remarque : Jointure externe gauche ou droite

Pour exprimer une jointure externe on se base sur la syntaxe INNER JOIN en utilisant à la place LEFT OUTER JOIN ou RIGHT OUTER JOIN.

Exemple : Jointure externe gauche

```
SELECT Num
FROM Avion LEFT OUTER JOIN Vol
ON Avion.Num=Vol.Num
```

Cette requête permet de sélectionner tous les avions, y compris ceux non affectés à un vol.

Remarque
Remarquons que "Avion LEFT OUTER JOIN Vol" est équivalent à "Vol RIGHT OUTER JOIN Avion" en terme de résultat.
Intuitivement, on préfère utiliser la jointure gauche pour sélectionner tous les tuple du côté N d'une relation 1:N, même si il ne sont pas référencés ; et la jointure droite pour pour sélectionner tous les tuples d'une relation 0:N, y compris ceux qui ne font pas de référence. Cette approche revient à toujours garder à gauche de l'expression "JOIN" la relation "principale", i.e. celle dont on veut tous les tuples, même s'ils ne référencent pas (ou ne sont pas référencés par) la relation "secondaire".

7. Opérateurs ensemblistes

Les opérateurs ensemblistes ne peuvent être exprimés à l'aide de l'instruction de sélection seule.

Syntaxe : Union

```
SELECT * FROM R1
UNION
SELECT * FROM R2
```

Syntaxe : Intersection

```
SELECT * FROM R1
INTERSECT
SELECT * FROM R2
```

Syntaxe : Différence

```
SELECT * FROM R1
EXCEPT
SELECT * FROM R2
```

Remarque
Les opérations INTERSECT et EXCEPT n'existe que dans la norme SQL2, et non dans la norme SQL1. Certain SGBD sont susceptibles de ne pas les implémenter.

8. Tri

On veut souvent que le résultat d'une requête soit trié en fonction des valeurs des propriétés des tuples de ce résultat.

Syntaxe : ORDER BY

```
SELECT <liste d'attributs projetés>
FROM <liste de relations>
WHERE <condition>
ORDER BY <liste ordonnée d'attributs>
```

Les tuples sont triés d'abord par le premier attribut spécifié dans la clause ORDER BY, puis en cas de doublons par le second, etc.

Remarque : Tri décroissant
Pour effectuer un tri décroissant on fait suivre l'attribut du mot clé "DESC".

Exemple

```
SELECT *
FROM Personne
ORDER BY Nom, Age DESC
```

9. Fonctions de calcul

Fonction de calcul

Une fonction de calcul s'applique à l'ensemble des valeurs d'une propriété d'une relation avec pour résultat la production d'une valeur atomique unique (entier, chaîne, date, etc).
Les cinq fonctions prédéfinies sont :

- **Count(Relation.Propriété)**

  Renvoie le nombre de valeurs non nulles d'une propriété pour tous les tuples d'une relation ;

- **Sum(Relation.Propriété)**

  Renvoie la somme des valeurs d'une propriété des tuples (numériques) d'une relation ;

- **Avg(Relation.Propriété)**

  Renvoie la moyenne des valeurs d'une propriété des tuples (numériques) d'une relation ;

- **Min(Relation.Propriété)**

  Renvoie la plus petite valeur d'une propriété parmi les tuples d'une relation .

- **Max(Relation.Propriété)**

  Renvoie la plus grande valeur d'une propriété parmi les tuples d'une relation.

Syntaxe

```
SELECT <liste de fonctions de calcul>
FROM <liste de relations>
WHERE <condition à appliquer avant calcul>
```

Exemple

```
SELECT Min(Age), Max(Age), Avg(Age)
FROM Personne
WHERE Qualification='Ingénieur'
```

Remarque : Utilisation de fonctions pour les agrégats

Dans le cas du calcul d'un agrégat, les fonctions peuvent être utilisées dans la claude HAVING ou dans la clause ORDER BY d'un tri. Les fonctions ne peuvent pas être utilisées dans la clause WHERE.

Remarque : Comptage d'une relation

Pour effectuer un comptage sur tous les tuples d'une relation, appliquer la fonction Count à un attribut de la clé primaire. En effet cet attribut étant non nul par définition, il assure que tous les tuples seront comptés.

10. Agrégats

Agrégat

Un agrégat est un partitionnement horizontal d'une table en sous-tables, en fonction des valeurs d'un ou plusieurs attributs de partitionnement, suivi de l'application d'une fonction de calcul à chaque attribut des sous-tables obtenues (Gardarin, 1999).

Syntaxe

```
SELECT <liste d'attributs de partionnement à projeter et de fonctions de calcul>
FROM <liste de relations>
WHERE <condition à appliquer avant calcul de l'agrégat>
GROUP BY <liste ordonnée d'attributs de partitionnement>
HAVING <condition sur les fonctions de calcul>
```

Exemple

```
SELECT Societe.Nom, AVG(Personne.Age)
FROM Personne, Societe
WHERE Personne.NomSoc = Societe.Nom
GROUP BY Societe.Nom
HAVING Count(Personne.NumSS) > 10
```

Cette requête calcul l'âge moyen du personnel pour chaque société comportant plus de 10 salariés.

Remarque : Restriction
Une restriction peut être appliquée avant calcul de l'agrégat, au niveau de la clause WHERE, portant ainsi sur la relation de départ, mais aussi après calcul de l'agrégat sur les résultats de ce dernier, au niveau de la clause HAVING.

Remarque : Projection
Si dans la clause SELECT, un attribut est projeté directement, sans qu'une fonction lui soit appliquée, alors il faut impérativement que cet attribut apparaisse dans la clause GROUP BY (car ce ne peut être qu'un attribut de partionnement).

Remarque : Fonctions de calcul sans partitionnement
Si une ou plusieurs fonctions de calcul sont appliquées sans partionnement, le résultat de la requête est un tuple unique.

Remarque : Intérêt de la clause GROUP BY
Pour que l'utilisation de la clause GROUP BY ait un sens, il faut qu'au moins une fonction de calcul soit utilisée, soit dans la clause SELECT, soit dans la clause HAVING.

Remarque : Contrôle imposé par quelques SGBDR
Notons que dans le cas de certains SGBDR (par exemple Oracle), l'ensemble des attributs de l'agrégation (clause GROUP BY) doivent être préalablement projetés (donc déclarés dans la clause SELECT).

11. Requêtes imbriquées

Il est possible d'imbriquer des requêtes les unes dans les autres pour procéduraliser les questions, et ainsi répondre à des questions plus complexes, voire impossibles, à écrire en algèbre relationnel classique.

Sous-requête
Requête incluse dans la clause WHERE ou FROM d'une autre requête.

- *Synonymes : Sous-question, Requête imbriquée.*

Syntaxe : Requêtes imbriquées par la clause WHERE

```
SELECT <projections>
FROM <relations>
WHERE <sous-requête>
```

Exemple

```
SELECT Nom
FROM Chercheur
WHERE Nom IN
   (SELECT Nom FROM Enseignant)
```

12. Sous-requête d'existence IN

Cette sous-requête permet de vérifier que la projection d'un tuple de la requête principale est présent dans la sous-requête.

Syntaxe

```
SELECT <projections>
FROM <relations>
WHERE (<projection d'un tuple>) IN
   (<requête imbriquée>)
```

La projection du tuple de la requête principale doit conduire à un schéma relationnel identique à celui de la requête imbriquée.

Exemple : Sous-requête IN à une colonne et plusieurs lignes

```
SELECT Chercheur.Nom
FROM Chercheur
WHERE Chercheur.Universite IN
   (SELECT Universite.Nom
   FROM Universite
   WHERE Universite.Ville='Paris')
```

Exemple : Sous-requête IN à plusieurs colonnes et plusieurs lignes

```
SELECT N°SS
FROM Chercheur
WHERE (Nom, Prenom, Age) IN
   (SELECT Nom, Prenom, Age
   FROM Enseignant)
```

Exemple : Imbrication multiple de requêtes

```
SELECT Nom
FROM Chercheur
WHERE Universite='Paris6' AND Nom IN
   (SELECT Nom
   FROM Enseignant
   WHERE Universite IN
      (SELECT Nom
      FROM Universite
      WHERE Ville='Paris'))
```

Remarque : Jointure par la sous-requête IN

La sous-requête IN est une troisième voie, avec les clauses WHERE et JOIN, pour réaliser des jointures entre relations. On préfèrera néanmoins éviter d'utiliser à cette unique fin cette version plus procédurale.

Remarque : NOT IN

On peut tester la non existence du tuple dans la sous requête en utilisant la clause NOT IN à la place de la clause IN.

13. Sous-requête d'existence EXISTS

Cette sous-requête permet de vérifier que la sous-requête contient au moins un tuple.

Syntaxe

```
SELECT <projections>
FROM <relations>
WHERE EXISTS
   (<requête imbriquée>)
```

La requête imbriquée faisant référence à des propriétés (éventuellement non projetées) de la requête principale.

Exemple

```
SELECT Chercheur.Nom
FROM Chercheur
WHERE EXISTS
   (SELECT *
   FROM Universite
   WHERE Universite.Nom=Chercheur.Universite)
```

Remarque : Projection dans la sous-requête

Puisque la sous-requête n'est destinée qu'à valider l'existence d'un tuple, il est inutile de procéder à une projection particulière pour cette sous-requête. On utilise donc en général la clause SELECT * pour une sous-requête avec une clause EXISTS.

Remarque : NOT EXISTS

On peut tester la non présence de tuple dans la sous-requête en utilisant la clause NOT EXISTS à la place de la clause EXISTS.

14. Sous-requête de comparaison ALL

Cette sous-requête permet de vérifier que les tuples de la requête principale vérifient bien une condition donnée avec tous les tuples de la sous-requête.

Syntaxe

```
SELECT <projections>
FROM <relations>
WHERE <propriété> <opérateur de comparaison> ALL
   (<requête imbriquée>)
```

La requête imbriquée renvoyant un tuple ne comportant qu'une propriété de même domaine que la propriété testée de la requête principale.

Exemple

```
SELECT Nom
FROM Chercheur
WHERE Age > ALL
   (SELECT Age
   FROM Etudiant)
```

15. Sous-requête de comparaison ANY

Cette sous-requête permet de vérifier que les tuples de la requête principale vérifie bien une condition donnée avec au moins un tuple de la sous-requête.

Syntaxe

```
SELECT <projections>
FROM <relations>
WHERE <propriété> <opérateur de comparaison> ANY
   (<requête imbriquée>)
```

La requête imbriquée renvoyant un tuple ne comportant qu'une propriété de même domaine que la propriété testée de la requête principale.

Exemple

```
SELECT Nom
FROM Chercheur
WHERE Age < ANY
   (SELECT Age
   FROM Etudiant)
```

Remarque : SOME

SOME peut être utilisé comme un synonyme de ANY.

16. Raffinement de questions dans la clause FROM

Explication

Il est possible de raffiner progressivement une requête en enchaînant les questions dans la clause FROM.

Syntaxe

```
SELECT ... FROM (SELECT ... FROM ... WHERE ...) WHERE ...)
```

Remarque

Il est possible d'enchaîner récursivement N questions.

Méthode

Cette extension est particulièrement utile pour les calculs d'aggégat après filtrage ou pour enchaîner les calculs d'aggrégat (par exemple pour faire la moyenne de sommes après regroupement).

Exemple : Enchaînement de calculs d'aggrégat

```
select avg(s) from (select sum(b) as s from t group by a)
```

17. Insertion de données

Le langage SQL fournit également des instructions pour ajouter des nouveaux tuples à une relation. Il offre ainsi une interface standard également pour ajouter des information dans une base de données.

Il existe deux moyens d'ajouter des données, soit par fourniture directe des valeurs des propriétés du tuple à ajouter, soit par sélection des tuples à ajouter dans une autre relation.

Syntaxe : Insertion directe de valeurs

```
INSERT INTO <Nom de la relation> (<Liste ordonnée des propriétés à valoriser>)
VALUES (<Liste ordonnée des valeurs à affecter aux propriétés spéficiées
ci-dessus>)
```

Exemple : Insertion directe de valeurs

```
INSERT INTO Virement (Date, Montant, Objet)
VALUES (14-07-1975, 1000, 'Prime de naissance')
```

Syntaxe : Insertion de valeurs par l'intermédiaire d'une sélection

```
INSERT INTO <Nom de la relation>, (<Liste ordonnée des propriétés à valoriser>)
SELECT ...
```

L'instruction SELECT projetant un nombre de propriétés identiques aux propriétés à valoriser.

Exemple : Insertion de valeurs par l'intermédiaire d'une sélection

```
INSERT INTO Credit (Date, Montant, Objet)
SELECT Date, Montant, 'Annulation de débit'
FROM Debit
WHERE Debit.Date = 25-12-2001
```

Dans cet exemple tous les débits effectués le 25 décembre 2001, sont recrédités pour le même montant (et à la même date), avec la mention annulation dans l'objet du crédit. Ceci pourrait typiquement réalisé en cas de débits errronés ce jour là.

Remarque

Les propriétés non valorisées sont affectées à la valeur null.

Il est possible de ne pas spécifier les propriétés à valoriser, dans ce cas, toutes les propriétés de la relation seront considérées, dans leur ordre de définition dans la relation (à n'utiliser que dans les cas les plus simples).

18. Mise à jour de données

Le langage SQL fournit une instruction pour modifier des tuples existants dans une relation.

Syntaxe : Mise à jour directe de valeurs

```
UPDATE <Nom de la relation>
SET <Liste d'affectation Propriété=Valeur>
WHERE <Condition pour filtrer les tuples à mettre à jour>
```

Exemple : Mise à jour directe de valeurs

```
UPDATE Compte
SET Monnaie='Euro'
WHERE Monnaie='Franc'
```

Exemple : Mise à jour par calcul sur l'ancienne valeur

```
UPDATE Compte
SET Total=Total * 6,55957
WHERE Monnaie='Euro'
```

19. Suppression de données

Le langage SQL fournit une instruction pour supprimer des tuples existants dans une relation.

Syntaxe

```
DELETE FROM <Nom de la relation>
WHERE <Condition pour filtrer les tuples à supprimer>
```

Exemple : Suppression de tous les tuples d'une relation

```
DELETE FROM FaussesFactures
```

Exemple : Suppression sélective

```
DELETE FROM FaussesFactures
WHERE Auteur='Moi'
```

Pour travailler les questions en SQL

"http://eric.univ-lyon2.fr/~jdarmont/tutoriel-sql/", Tutoriel SQL, **DARMONT J,** janvier, 2004.

Un excellent exerciseur permettant de poser des questions en SQL à une BD en temps réel.

18 questions à faire absolument.

Cours général

IV

Normalisation, SQL LDD et LCD

La théorie de la normalisation relationnelle est très importante pour la conception de BD, dans la mesure où elle donne le cadre théorique pour la gestion de la redondance, et dans la mesure où une bonne maîtrise de la redondance est un aspect majeur de cette conception.
Après avoir expliqué la théorie de la normalisation et donné les outils méthodologiques pour la gestion de la redondance, ce chapitre terminera la description du langage SQL par la description des instructions des langages LDD et LCD, qui composent avec le LMD les trois parties du langage SQL.

Chapitre A. La normalisation

Objectifs pédagogiques
Comprendre la problématique de la redondance et des dépendances fonctionnelles.
Savoir créer des schémas relationnels en troisième forme normale.

Section A1. Théorie de la normalisation relationnelle

1. Les problèmes soulevés par une mauvaise modélisation

Attention
Il y a toujours plusieurs façons de modéliser conceptuellement un problème, certaines sont bonnes et d'autres mauvaises. C'est l'expertise de l'ingénieur en charge de la modélisation, à travers son expérience accumulée et sa capacité à traduire le problème posé, qui permet d'obtenir de bons modèles conceptuels.

S'il est difficile de définir un bon modèle conceptuel, on peut par contre poser qu'un bon modèle logique relationnel est un modèle où la redondance est contrôlée. On peut alors poser qu'un bon modèle conceptuel est un modèle conceptuel qui conduit à un bon modèle relationnel, après application des règles de passage E-A ou UML vers relationnel. Mais on ne sait pas pour autant le critiquer avant ce passage, autrement qu'à travers l'oeil d'un expert.

A défaut de disposer d'outils systématiques pour obtenir de bons modèles conceptuels, on cherche donc à critiquer les modèles relationnels obtenus. La théorie de la normalisation est une théorie qui permet de critiquer, puis d'optimiser, des modèles relationnels, de façon à en contrôler la redondance.

Exemple : Un mauvais modèle relationnel
Imaginons que nous souhaitions représenter des personnes, identifiées par leur numéro de sécurité sociale, caractérisées par leur nom, leur prénom, ainsi que les véhicule qu'elles ont acheté, pour un certain prix et à une certaine date, sachant qu'un véhicule est caractérisé par un type, une marque et une puissance. On peut aboutir à la représentation relationnelle suivante :

```
Personne(NSS, Nom, Prénom, Marque, Type, Puiss, Date, Prix)
```

Imaginons que cette relation soit remplie par les données suivantes :

NSS	Nom	Prénom	Marque	Type	Puiss	Date	Prix
16607...	Dupont	Paul	Renault	Clio	5	1/1/96	60000
16607...	Dupont	Paul	Peugeot	504	7	2/7/75	47300
24908...	Martin	Marie	Peugeot	504	7	1/10/89	54900
15405...	Durand	Olivier	Peugeot	504	7	8/8/90	12000
15405...	Durand	Olivier	Renault	Clio	5	7/6/98	65000
15405...	Durand	Olivier	BMW	520	10	4/5/2001	98000

▲ TAB. 13 : RELATION PERSONNE

On peut alors se rendre compte que des redondances sont présentes, et l'on sait que ces redondances conduiront à des problèmes de contrôle de la cohérence de l'information (erreur dans la saisie d'un numéro de sécurité sociale), de mise à jour (changement de nom à reporter dans de multiples tuples), de perte d'information lors de la suppression de données (disparition des informations concernant un type de véhicule) et de difficulté à représenter certaines informations (un type de véhicule sans propriétaire).

DELMAL P, "SQL2 SQL3, applications à Oracle", De Boeck Université, 2001.

On conseillera de lire le chapitre 2 (pages 42 à 49) qui propose une très bonne démonstration par l'exemple des problèmes posés par une mauvaise modélisation relationnelle.

2. Principes de la normalisation

Fondamentaux
La théorie de la normalisation est une théorie destinée à concevoir un bon schéma d'une BD sans redondance d'information et sans risques d'anomalie de mise à jour. Elle a été introduite dès l'origine dans le modèle relationnel (Codd, 1970) .

La théorie de la normalisation est fondée sur deux concepts principaux :

- **Les dépendances fonctionnelles**

  Elles traduisent des contraintes sur les données.

- **Les formes normales**

  Elles définissent des relations bien conçues.

La mise en oeuvre de la normalisation est fondée sur la décomposition progressive des relations jusqu'à obtenir des relations normalisées.

3. Dépendance fonctionnelle

Dépendance fonctionnelle
Soient R(A1, A2, ... , An) un schéma de relation, X et Y des sous-ensembles de A1, A2, ... , An. On dit que X détermine Y, ou que Y dépend fonctionnellement de X, si est seulement s'il existe une fonction qui à partir de toute valeur de X détermine une valeur unique de Y.
Plus formellement on pose que X détermine Y pour une relation R ssi quelle que soit l'instance r de R, alors pour tous tuples t1 et t2 de r on a :
Projection (t1,X) = Projection (t2,X) ⇒ Projection (t1,Y) = Projection (t2,Y)

Syntaxe
Si X détermine Y, on note : X→ Y

Exemple
Soit la relation R suivante :

```
Personne(NSS, Nom, Prénom, Marque, Type, Puiss, Date, Prix)
```

On peut poser les exemples de DF [Dépendance Fonctionnelle] suivants :

- NSS→ Nom
- NSS→ Prénom
- Type→ Marque
- Type→ Puiss
- (NSS, Type, Date)→ Prix
- etc.

Remarque : Comment trouver les DF ?
Une DF est définie sur l'intention du schéma et non son extension. Une DF traduit une certaine perception de la réalité. Ainsi la DF (NSS, Type, Date)→ Prix signifie que personne n'achète deux voitures du même type à la même date.
La seule manière de déterminer une DF est donc de regarder soigneusement ce que signifient les attributs et de trouver les contraintes qui les lient dans le monde réel.

Remarque : Pourquoi trouver les DF ?
Les DF font partie du schéma d'une BD, en conséquence, elles doivent être déclarées par les administrateurs de la BD et être contrôlées par le SGBD.
De plus l'identification des DF est la base indispensable pour déterminer dans quelle forme normale est une relation et comment en diminuer la redondance.

4. Les axiomes d'Armstrong

Les DF obéissent à des propriétés mathématiques particulières, dites axiomes d'Armstrong.

Réflexivité
Tout groupe d'attributs se détermine lui même et détermine chacun de ses attributs (ou sous groupe de ses attributs).
Soient X et Y des attributs :
XY→ XY et XY→ X et XY→ Y

Augmentation
Si un attribut X détermine un attribut Y, alors tout groupe composé de X enrichi avec d'autres attributs détermine un groupe composé de Y et enrichi des mêmes autres attributs.
Soient X, Y et Z des attributs :
X→ Y ⇒ XZ→ YZ

Transitivité
Si un attribut X détermine un attribut Y et que cet attribut Y détermine un autre attribut Z, alors X détermine Z.
Soient X, Y et Z des attributs :
X→ Y et Y→ Z ⇒ X→ Z

5. Autres propriétés déduites des axiomes d'Armstrong

A partir des axiomes d'Amstrong, on peut déduire un certain nombre de propriétés supplémentaires.

Pseudo-transitivité

Si un attribut X détermine un autre attribut Y, et que Y appartient à un groupe G qui détermine un troisième attribut Z, alors le groupe G' obtenu en substituant Y par X dans G détermine également Z.
Soient, W, X, Y et Z des attributs :
X→ Y et WY→ Z ⇒ WX→ Z
Cette propriété est déduite de l'augmentation et de la réflexivité :
X→ Y et WY→ Z ⇒ WX→ WY et WY→ Z ⇒ WX→ Z

Union

Si un attribut détermine plusieurs autres attributs, alors il détermine tout groupe composé de ces attributs.
Soient X, Y et Z des attributs :
X→ Y et X→ Z ⇒ X→ YZ
Cette propriété est déduite de la réflexivité, de l'augmentation et de la transitivité :
X→ Y et X→ Z ⇒ X→ XX et XX→ XY et YX→ YZ ⇒ X→ YZ

Décomposition

Si un attribut détermine un groupe d'attribut, alors il détermine chacun des attributs de ce groupe pris individuellement.
Soient X, Y et Z des attributs :
X→ YZ ⇒ X→ Z et X→ Y
Cette propriété est déduite de la réflexivité et de la transitivité :
X→ YZ ⇒ X→ YZ et YZ→ Z ⇒ X→ Z

6. DF élémentaire

Dépendance fonctionnelle élémentaire

Soit G un groupe d'attributs et A un attribut, une DF G→ A est élémentaire si A n'est pas inclu dans G et qu'il n'existe pas d'attribut A' de G qui détermine A.

Exemple : DF élémentaires

- AB→ C est élémentaire si ni A, ni B pris individuellement ne déterminent C.
- Nom, DateNaissance, LieuNaissance→ Prénom est élémentaire.

Exemple : DF non élémentaires

- AB→ A n'est pas élémentaire car A est inclu dans AB.
- AB→ CB n'est pas élémentaire car CB n'est pas un attribut, mais un groupe d'attributs.
- N°SS→ Nom, Prénom n'est pas élémentaire.

Remarque

On peut toujours réécrire un ensemble de DF en un ensemble de DFE [Dépendance Fonctionnelle Elémentaire], en supprimant les DF triviales obtenues par réflexivité et en décomposant les DF à partie droite non atomique en plusieurs DFE.

Exemple : Réécriture de DF en DFE

On peut réécrire les DF non élémentaires de l'exemple précédent en les décomposant DFE :

- AB→ A n'est pas considérée car c'est une DF triviale obtenu par réfléxivité.
- AB→ CB est décomposée en AB→ C et AB→ B, et AB→ B n'est plus considérée car triviale.
- N°SS→ Nom, Prénom est décomposée en N°SS→ Nom et N°SS→ Prénom.

7. Notion de fermeture transitive des DFE

Fermeture transitive

On appelle fermeture transitive F+ d'un ensemble F de DFE, l'ensemble de toutes les DFE qui peuvent être composées par transitivité à partir des DFE de F.

Exemple
Soit l'ensemble F = {A→ B, B→ C, B→ D, A→ E}.

La fermeture transitive de F est F+ = { A→ B, B→ C, B→ D, A→ E, A→ C, A→ D }

8. Notion de couverture minimale des DFE

Couverture minimale
La couverture minimale d'un ensemble de DFE est un sous-ensemble minimum des DFE permettant de générer toutes les autres DFE.

- *Synonyme : Famille génératrice.*

Remarque
Tout ensemble de DFE (et donc tout ensemble de DF) admet au moins une couverture minimale (et en pratique souvent plusieurs).

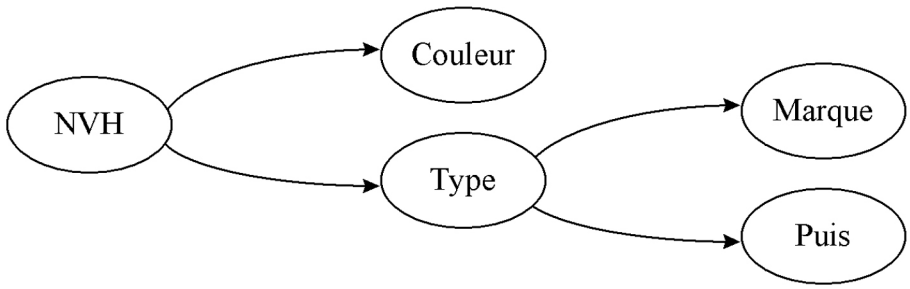
Exemple
L'ensemble F = {A→ B, A→ C, B→ C, C→ B} admet les deux couvertures minimales :

CM1 = {A→ C, B→ C, C→ B} et CM2 = {A→ B, B→ C, C→ B}

9. Notion de graphe des DFE

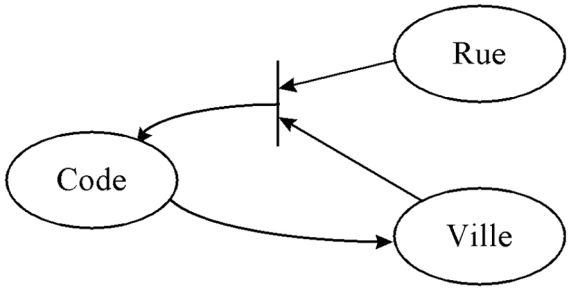
On peut représenter un ensemble de DFE par un graphe orienté (ou plus précisément un réseau de Pétri), tel que les noeuds sont les attributs et les arcs les DFE (avec un seul attribut en destination de chaque arc et éventuellement plusieurs en source).

Exemple : Relation Voiture
Soit la relation Voiture(NVH, Marque, Type, Puis, Couleur) avec l'ensemble des DF F = {NVH→ Type, Type→ Marque, Type→ Puis, NVH→ Couleur}. On peut réprésenter F par le graphe ci-dessous :



▲ IMG. 27 : GRAPHE DES DFE DE LA RELATION VOITURE

Exemple : Relation CodePostal
Soit la relation CodePostal(Code, Ville, Rue) avec l'ensemble des DF F={Code→ Ville, (Ville,Rue)→ Code}. On peut réprésenter F par le graphe ci-dessous :



▲ IMG. 28 : GRAPHE DES DFE DE LA RELATION CODEPOSTAL

10. Définition formelle d'une clé

Clé
Soient une relation R(A1,A2,...,An) et K un sous-ensemble de A1,A2,... ,An. K est une clé de R si et seulement si K→ A1,A2,...,An et il n'existe pas X inclu dans K tel que X→ A1,A2,...,An.
Une clé est donc un ensemble minimum d'attributs d'une relation qui détermine tous les autres.

Remarque : Clés candidates et clé primaire
Si une relation comporte plusieurs clés, chacun est dite clé candidate et l'on en choisit une en particulier pour être la clé primaire.

Remarque
Toute clé candidate détermine les autres clés candidates, puisque qu'une clé détermine tous les attributs de la relation.

11. Principe de la décomposition

Décomposition
L'objectif de la décomposition est de "casser" une relation en relations plus petites afin d'en éliminer les redondances et sans perdre d'information.
La décomposition d'un schéma de relation R(A1,A2,...,An) est le processus de remplacement de ce schéma par une collection de schémas R1,R2,...,Rn telle qu'il est possible de reconstruire R par des opérations relationnelles de jointure sur R1,R2,...,Rn.

Décomposition préservant les DF
Une décomposition d'une relation R en relations R1,R2,...Rn préserve les DF si la fermeture transitive F+ des DF de R est la même que celle de l'union des fermetures transitives des DF de R1,R2,...,Rn.

Exemple : Décomposition préservant les DF d'une relation Voiture
Soit la relation Voiture(Numéro,Marque,Type,Puissance,Couleur) avec la fermeture transitive suivante :

- Numéro→ Marque
- Numéro→ Type
- Numéro→ Puissance
- Numéro→ Couleur
- Type→ Marque
- Type→ Puissance

On peut décomposer Voiture en préservant les DF en deux relations R1(Numéro,Type,Couleur) et R2(Type,Puissance,Marque).

12. Formes normales

Les formes normales ont pour objectif de définir la décomposition des schémas relationnels, tout en préservant les DF et sans perdre d'informations, afin de représenter les objets et associations canoniques du monde réel de façon non redondante.

On peut rescencer les 6 formes normales suivantes, de moins en moins redondantes :
- la première forme normale
- la deuxième forme normale
- la troisième forme normale
- la forme normale de Boyce-Codd
- la quatrième forme normale
- la cinquième forme normale

La troisième forme normale est généralement reconnue comme étant la plus importante à respecter.

13. Première forme normale

1NF
Une relation est en 1NF [Première Forme Normale] si elle possède une clé et si tous ses attributs sont atomiques.

Attribut atomique
Un attribut est atomique si il ne contient qu'une seule valeur pour un tuple donné, et donc s'il ne regroupe pas un ensemble de plusieurs valeurs.

Exemple : Avoir plusieurs métiers
Soit la relation Personne instanciée par deux tuples :

```
Personne(Nom, Profession)
```

```
(Dupont, Géomètre)
(Durand, Ingénieur-Professeur)
```

La relation n'est pas en 1NF, car l'attribut Profession peut contenir plusieurs valeurs.

Pour que la relation soit en 1NF, on pourrait par exemple ajouter Profession à la clé et faire apparaître deux tuples pour Durand, on obtiendrait :

```
Personne(Nom, Profession)
```

```
(Dupont, Géomètre)
(Durand, Ingénieur)
(Durand, Professeur)
```

Une autre solution aurait été d'ajouter un attribut ProfessionSecondaire. On obtiendrait ainsi :

```
Personne(Nom, Profession, ProfessionSecondaire)
```

```
(Dupont, Géomètre, Null)
(Durand, Ingénieur, Professeur)
```

Remarque : Relativité de la notion d'atomicité
L'atomicité d'un attribut est souvent relative : on peut décider qu'un attribut contenant une date n'est pas atomique (et que le jour, le mois et l'année constituent chacun une valeur), ou bien que l'attribut est de domaine date et donc qu'il est atomique.

14. Deuxième forme normale

La deuxième forme normale permet d'éliminer les dépendances entre des parties de clé et des attributs n'appartenant pas à une clé.

2NF
Une relation est en 2NF [Deuxième Forme Normale] si elle est en 1NF et si tout attribut qui n'est pas dans une clé ne dépend pas d'une partie seulement d'une clé. C'est à dire encore que toutes les DF issues d'une clé sont élémentaires.

Exemple : Echelle de salaire
Soit la relation Personne :

```
Personne(Nom, Profession, Salaire)
```

Soit les DF suivantes sur cette relation :
- Nom,Profession→ Salaire
- Profession→ Salaire

On note alors que la première DF est issue de la clé et qu'elle n'est pas élémentaire (puisque Profession détermine Salaire) et donc que le schéma n'est pas en 2NF.
Pour avoir un schéma relationnel en 2NF, il faut alors décomposer Personne en deux relations :

```
Personne(Nom, Profession)
Profession(Profession, Salaire)
```

On remarque que ce schéma est en 2NF (puisque Salaire dépend maintenant fonctionnellement d'une clé et non plus d'une partie de clé).
On remarque aussi que la décomposition a préservé les DF, puisque nous avons à présent :
- Profession→ Salaire (DF de la relation Profession)
- Nom,Profession→ Profession (par Réflexivité)
- Nom,Profession→ Salaire (par Transitivité)

Remarque
La définition de la 2NF doit être vérifiée pour toutes les clés candidates et non seulement la clé primaire (dans le cas où il y a plusieurs clés).

Remarque
Si toutes les clés d'une relation ne contiennent qu'un unique attribut, et que la relation est en 1NF, alors la relation est en 2NF.

15. Troisième forme normale

La troisième forme normale permet d'éliminer les dépendances entre les attributs n'appartenant pas à une clé.

3NF
Une relation est en 3NF [Troisième Forme Normale] si elle est en 2NF et si tout attribut n'appartenant pas à une clé ne dépend pas d'un autre attribut n'appartenant pas à une clé. C'est à dire encore que toutes les DFE vers des attributs n'appartenant pas à une clé, sont issues d'une clé.

Clé candidate
La définition concerne toutes les clés candidates et non uniquement la clé primaire(Celko, 2000) (p.27).

Exemple : Echelle de salaire et de prime
Soit la relation Profession :

```
Profession(Profession, Salaire, Prime)
```

Soit les DF suivantes sur cette relation :
- Profession→ Salaire
- Profession→ Prime
- Salaire→ Prime

Cette relation n'est pas en 3NF car Salaire, qui n'est pas une clé, détermine Prime.
Pour avoir un schéma relationnel en 3NF, il faut décomposer Profession :

```
Profession(Profession, Salaire)
Salaire(Salaire, Prime)
```

Ce schéma est en 3NF, car Prime est maintenant déterminé par une clé.
On remarque que cette décomposition préserve les DF, car par transitivité, Profession détermine Salaire qui détermine Prime, et donc Profession détermine toujours Prime.

Remarque : 3NF et 2NF
Une relation en 3NF est forcément en 2NF car :

- Toutes les DFE vers des attributs n'appartenant pas à une clé sont issues d'une clé, ce qui implique qu'il n'existe pas de DFE, issues d'une partie de clé vers un attribut qui n'appartient pas à une clé.
- Il ne peut pas non plus exister de DFE issues d'une partie de clé vers un attribut appartenant à une clé, par définition de ce qu'une clé est un ensemble minimum.

On n'en conclut qu'il ne peut exister de DFE, donc a fortiori pas de DF, issues d'une partie d'une clé, et donc que toutes les DF issues d'une clé sont élémentaires.

Fondamentaux
Il est souhaitable que les relations logiques soient en 3NF. En effet, il existe toujours une décomposition sans perte d'information et préservant les DF d'un schéma en 3NF. Si les formes normales suivantes (BCNF [Forme Normale de Boyce-Codd], 4NF [Quatrième Forme Normale] et 5NF [Cinquième Forme Normale]) assurent un niveau de redondance encore plus faible, la décomposition permettant de les atteindre ne préserve plus les DF.

Remarque : Limite de la 3NF
Une relation en 3NF permet des dépendances entre des attributs n'appartenant pas à une clé vers des parties de clé.

16. Forme normale de Boyce-Codd

La forme normale de Boyce-Codd permet d'éliminer les dépendances entre les attributs n'appartenant pas à une clé vers les parties de clé.

BCNF
Une relation est en BCNF si elle est en 3NF et si tout attribut qui n'appartient pas à une clé n'est pas source d'une DF vers une partie d'une clé. C'est à dire que les seules DFE existantes sont celles dans lesquelles une clé détermine un attribut.

Exemple : Employés
Soit la relation Personne :

```
Personne(N°SS, Pays, Nom, Région)
```

Soit les DF suivantes sur cette relation :

- N°SS,Pays→ Nom

  N°SS,Pays→ Région

  Région→ Pays

Il existe une DFE qui n'est pas issue d'une clé et qui détermine un attribut appartenant à une clé. Cette relation est en 3NF, mais pas en BCNF (car en BCNF toutes les DFE sont issues d'une clé).
Pour avoir un schéma relationnel en BCNF, il faut décomposer Personne :

```
Personne(N°SS, Région, Nom)
Région(Region, Pays)
```

Remarquons que les DF n'ont pas été préservées par la décomposition puisque N°SS et Pays ne déterminent plus Région.

Remarque : Simplicité
La BCNF est la forme normale la plus facile à appréhender intuitivement et formellement, puisque les seules DFE existantes sont de la forme K→ A où K est une clé.

Non préservation des DF
Une décomposition en BCNF ne préserve en général pas les DF.

* *
*

La normalisation permet de décomposer un schéma relationnel afin d'obtenir des relations non redondantes.
La 3NF est souhaitable car toujours possible à obtenir, sans perte d'information et sans perte de DF. La BCNF est également indiquée, car elle est un peu plus puissante, et plutôt plus simple que la 3NF.
La BCNF n'est pas encore suffisante pour éliminer toutes les redondances. Il existe pour cela les 4NF et 5NF qui ne sont pas abordées dans ce cours. Notons également que les cas de non-4NF et de non-5NF sont assez rares dans la réalité.

Pour continuer ...

DELMAL P, "SQL2 SQL3, applications à Oracle", De Boeck Université, 2001.

Une très bonne démonstration exemplifiée qu'il y a des bons et mauvais schémas relationnels (chapitre 2, pages 43-49).

Une description claire des formes normales, rendue simple et pratique grâce à des exemples représentatifs (chapitre 2).

Chapitre B. Définition et contrôle de données

Objectifs pédagogiques
Maîtriser les bases du SQL pour créer des tables, attribuer des droits et entrer, modifier et effacer des données dans les tables.

Bref aperçu

SQL est un langage déclaratif destiné aux SGBD et plus particulièrement aux SGBDR. Ce langage est défini par une norme ISO datant de 1986 pour SQL1 et 1992 pour SQL2[La norme SQL3 en 1999 étend SQL aux SGBD Relationnels-Objets]. SQL n'est pas a proprement parlé un langage de programmation, mais plutôt une interface standard pour accéder aux bases de données.

Il est composé de trois sous ensembles :
- Le LDD pour créer et supprimer des objets dans la base de données (tables, contraintes d'intégrité, vues, etc.).
- Le LCD pour gérer les droits sur les objets de la base (création des utilisateurs et affectation de leurs droits).
- Le LMD pour la recherche, l'insertion, la mise à jour et la suppression de données.

  Le LMD est basé sur les opérateurs relationnels, auxquels sont ajoutés des fonctions de calcul d'agrégats et des instructions pour réaliser les opérations d'insertion, mise à jour et suppression.

Section B1. Le LDD de SQL

Le LDD permet de créer les objets composant une BD de façon déclarative. Il permet notamment la définition des schémas des relations, la définition des contraintes d'intégrité, la définition de vues relationnelles.

1. Types de données

Un attribut d'une relation est défini pour un certain domaine. On peut également dire qu'il est d'un type particulier. Les types de données disponibles en SQL varient d'un SGBD à l'autre, on peut néanmoins citer un certain nombre de types standards que l'on retrouve dans tous les SGBD.

1.1. Les types numériques

- **Les nombres entiers**

  INTEGER(X), où X est optionnel et désigne le nombre de chiffres maximum pouvant composer le nombre. Il existe également un certain nombre de variantes permettant de définir des entiers plus ou moins volumineux, tels que TINYINT, SMALLINT ou LONGINT.

- **Les nombres décimaux**

  DECIMAL(X,Y), où X et Y sont optionnels et désignent respectivement le nombre de chiffres maximum pouvant composer le nombre avant et après la virgule. NUMERIC est également utilisé de façon équivalente.

- **Les nombres à virgule flottante**

  REAL(X,Y), avec X et Y optionnels et définissant le nombre de chiffres avant et après la virgule.

  Il existe également un certain nombre de variantes permettant de définir une précision plus grande, telles que DOUBLE.

1.2. Les types chaîne de caractères

On distingue principalement les types CHAR(X) et VARCHAR(X), où X et obligatoire et désigne la longueur de la chaîne. CHAR définit des chaînes de longueur fixe (complétée à droites par des espaces, si la longeur est inférieure à X) et VARCHAR des chaînes de longueurs variables. CHAR et VARCHAR sont généralement limités à 255 caractères. La plupart des SGBD proposent des types, tels que TEXT ou CLOB (Character Long Object), pour représenter des chaînes de caractères longues, jusqu'à 65000 caractères par exemple.

1.3. Les types date

Les types date dont introduits avec la norme SQL2. On distingue le type DATE qui représente une date selon un format de type "AAAA-MM-JJ" et le types DATETIME qui représente une date avec un horaire, dans un format tel que "AAAA-MM-JJ HH:MM:SS". Il existe également des restrictions de ces types, tels que YEAR, TIME, etc.

1.4. Les autres types

En fonction du SGBD, il peut exister de nombreux autres types. On peut citer par exemple les types monétaires pour représenter des décimaux associés à une monnaie, les types ENUM pour représenter des énumérations, les types BLOB (pour Binary Long Oject) pour représenter des données binaires tels que des documents multimédia (images bitmap, vidéo, etc.).

1.5. Le type absence de valeur

L'absence de valeur, représentée par la valeur NULL, est une information fondamentale en SQL, qu'il ne faut pas confondre avec la chaîne espace de caractère où bien la valeur 0. Il ne s'agit pas d'un type à proprement parler, mais d'une valeur possible dans tous les types.

2. Création de tables

La création de table est le fondement de la création d'une base de données en SQL.

Création de table
La création de table est la définition d'un schéma de relation en intension, par la spécification de tous les attributs le composant avec leurs domaines respectifs.

Syntaxe

```
CREATE TABLE <nom de table> (
   <nom colonne1> <type colonne1>,
   <nom colonne2> <type colonne2>,
   ...
   <nom colonneN> <type colonneN>,
);
```

Exemple

```
CREATE TABLE Personne (
   Nom VARCHAR(25),
   Prenom VARCHAR(25),
   Age INTEGER(3)
);
```

Contrainte d'intégrité

La définition des attributs n'est pas suffisante pour définir un schéma relationnel, il faut lui adjoindre la définition de *contraintes d'intégrité*, qui permette de poser les notions de clé, d'intégrité référentielle, de restriction de domaines, etc.

3. Contraintes d'intégrité

Contraintes d'intégrité

Une contrainte d'intégrité est une règle qui définit la cohérence d'une donnée ou d'un ensemble de données de la BD.
Il existe deux types de contraintes :

- sur une colonne unique,
- ou sur une table lorsque la contrainte porte sur une ou plusieurs colonnes.

Les contraintes sont définies au moment de la création des tables.
Les contraintes d'intégrité sur une colonne sont :

- PRIMARY KEY : définit l'attribut comme la clé primaire
- NOT NULL : interdit l'absence de valeur pour l'attribut
- UNIQUE : interdit que deux tuples de la relation aient la même valeur pour l'attribut.
- REFERENCES <nom table> (<nom colonnes>) : contrôle l'intégrité référentielle entre l'attribut et la table et ses colonnes spécifiées
- CHECK (<condition>) : contrôle la validité de la valeur de l'attribut spécifié dans la condition dans le cadre d'une restriction de domaine

Les contraintes d'intégrité sur une table sont :

- PRIMARY KEY (<liste d'attibuts>) : définit les attributs de la liste comme la clé primaire
- UNIQUE (<liste d'attibuts>) : interdit que deux tuples de la relation aient les mêmes valeurs pour l'ensemble des attributs de la liste.
- FOREIGN KEY (<liste d'attibuts>) REFERENCES <nom table>(<nom colonnes>) : contrôle l'intégrité référentielle entre les attributs de la liste et la table et ses colonnes spécifiées
- CHECK (<condition>) : contrôle la validité de la valeur des attributs spécifiés dans la condition dans le cadre d'une restriction de domaine

Syntaxe

```
CREATE TABLE <nom de table> (
   <nom colonne1> <type colonne1> <contraintes colonne1>,
   <nom colonne2> <type colonne2> <contraintes colonne2>,
   ...
   <nom colonneN> <type colonneN> <contraintes colonneN>,
   <contraintes de table>
);
```

Remarque : Clé candidate
La contrainte UNIQUE NOT NULL sur un attribut ou un groupe d'attributs définit une clé candidate non primaire.

Remarque
Les contraintes sur une colonne et sur une table peuvent être combinées dans la définition d'un même schéma de relation.
Une contrainte sur une colonne peut toujours être remplacée par une contrainte sur une table.

Exemple

```
CREATE TABLE Personne (
   N°SS CHAR(13) PRIMARY KEY,
   Nom VARCHAR(25) NOT NULL,
   Prenom VARCHAR(25) NOT NULL,
   Age INTEGER(3) CHECK (Age BETWEEN 18 AND 65),
   Mariage CHAR(13) REFERENCES Personne(N°SS),
   Codepostal INTEGER(5),
   Pays VARCHAR(50),
   UNIQUE (Nom, Prenom),
   FOREIGN KEY (Codepostal, Pays) REFERENCES Adresse (CP, Pays)
);


CREATE TABLE Adresse (
   CP INTEGER(5) NOT NULL,
   Pays VARCHAR(50) NOT NULL,
   Initiale CHAR(1) CHECK (Initiale = LEFT(Pays, 1)),
   PRIMARY KEY (CP, Pays)
);
```

Dans la définition de schéma précédente on a posé les contraintes suivantes :

- La clé primaire de Personne est N°SS et la clé primaire de Adresse est (CP, Pays).
- Nom, Prénom ne peuvent pas être null et (Nom, Prénom) est une clé.
- Age doit être compris entre 18 et 65 et Initiale doit être la première lettre de Pays (avec la fonction LEFT qui renvoie la sous chaîne à gauche de la chaîne passée en premier argument, sur le nombre de caractères passés en second argument)
- Mariage est clé étrangère vers Personne et (Codepostal, Pays) est une clé étrangère vers Adresse.

4. Création de vues

Vue
Une vue est une définition logique d'une relation, sans stockage de données, obtenue par interrogation d'une ou plusieurs tables de la BD. Une vue peut donc être perçue comme une fenêtre dynamique sur les données, ou encore une requête stockée (mais dont seule la définition est stockée, pas le résultat, qui reste calculé dynamiquement).

Une vue permet d'implémenter le concept de schéma externe d'un modèle conceptuel.

- *Synonymes : Relation dérivée, Table virtuelle calculée.*

Syntaxe

```
CREATE VIEW <nom de vue> <nom des colonnes>
AS <spécification de question>
```

La spécification d'une question se fait en utilisant le LMD.
Le nombre de colonnes nommées doit être égal au nombre de colonnes renvoyées par la question spécifiée. Le nom des colonnes est optionnel, s'il n'est pas spécifié, c'est le nom des colonnes telle qu'elles sont renvoyées par la question, qui sera utilisé.

Exemple

```
CREATE VIEW Employe (Id, Nom)
AS
   SELECT N°SS, Nom
   FROM Personne
```

La vue Employe est ici une projection de la relation Personne sur les attributs N°SS et Nom, renommés respectivement Id et Nom.

Remarque : Vue en lecture et vue en écriture

Une vue est toujours disponible en lecture, à condition que l'utilisateur ait les droits spécifiés grâce au LCD. Une vue peut également être disponible en écriture dans certains cas, que l'on peut restreindre aux cas où la question ne porte que sur une seule table (même si dans certains cas, il est possible de modifier une vue issue de plusieurs tables).

Dans le cas où une vue est destinée à être utilisée pour modifier des données, il est possible d'ajouter la clause "WITH CHECK OPTION" après la spécification de question, pour préciser que les données modifiées ou ajoutées doivent effectivement appartenir à la vue.

Remarque : Vue sur une vue

Une vue peut avoir comme source une autre vue.

5. Supression d'objets

Il est possible de supprimer des objets de la BD, tels que les tables ou les vues.

Syntaxe

```
DROP <type objet> <nom objet>
```

Exemple

```
DROP TABLE Personne;
DROP VIEW Employe;
```

6. Modification de tables

L'instruction ALTER TABLE permet de modifier la définition d'une table (colonnes ou contraintes) préalablement créée.

Cette commande absente de SQL-89 est normalisée dans SQL-92

Syntaxe : Ajout de colonne

```
ALTER TABLE <nom de table>
   ADD (<définition de colonne>);
```

Syntaxe : Suppression de colonne

```
ALTER TABLE <nom de table>
   DROP (<nom de colonne>);
```

Syntaxe : Modification de colonne

```
ALTER TABLE <nom de table>
   MODIFY (<redéfinition de colonne>);
```

Syntaxe : Ajout de contrainte

```
ALTER TABLE <nom de table>
   ADD (<définition de contrainte de table>);
```

Remarque : Modification de table sans donnée sans la commande ALTER

Pour modifier une table ne contenant pas encore de donnée, la commande ALTER n'est pas indispensable, l'on peut supprimer la table à modifier (DROP) et la recréer telle qu'on la souhaite. Notons néanmoins que si la table est référencée par des clauses FOREIGN KEY, cette suppression sera plus compliquée, car il faudra également suprimer et recréer les tables référençantes (ce qui ce complique encore si ces dernières contiennent des données).

Remarque : Modification de table avec données sans la commande ALTER
Pour modifier une table contenant des données, la commande ALTER n'est pas absolument indispensable. On peut en effet :

1. Copier les données dans une table temporaire de même schéma que la table à modifier
2. Supprimer et recréer la table à modifier avec le nouveau schéma
3. Copier les données depuis la table temporaire vers la table modifiée

7. Exemple de modifications de tables

Table initiale
Soit une table intiale telle que définie ci-après.

```
create table t_personnes (
   pk_n number (4),
   nom varchar(50),
   prenom varchar (50),
   PRIMARY KEY (pk_n)
);
```

Modifications
On décide d'apporter les aménagements suivants à la table : on passe la taille du champ "nom" de 50 à 255 caractères maximum, on définit "nom" comme UNIQUE et on supprime le champ "prenom".

```
alter table t_personnes
   modify (nom varchar(255));


alter table t_personnes
   add (UNIQUE (nom));


alter table t_personnes
   drop (prenom);
```

Table finale
La table obtenue après modification est identique à la table qui aurait été définie directement telle que ci-après.

```
create table t_personnes (
   pk_n number (4),
   nom varchar(255),
   PRIMARY KEY (pk_n),
   UNIQUE (nom)
);
```

Section B2. Le LCD de SQL

Le LCD permet de créer les utilisateurs et de définir leurs droits sur les objets de la BD de façon déclarative. Il permet notamment l'attribution et la révocation de droits à des utilisateurs, sur l'ensemble des bases du SGBD, sur une BD en particulier, sur des relations d'une BD, voire sur certains attributs seulement d'une relation.

1. Attribution de droits

SQL propose une commande pour attribuer des droits à des utilisateurs sur des tables.

Syntaxe

```
GRANT <liste de droits> ON <nom table> TO <utilisateur> [WITH GRANT OPTION]
```

Les droits disponibles renvoient directement aux instructions SQL que l'utilisateur peut exécuter :

- SELECT
- INSERT
- DELETE
- UPDATE
- ALTER

De plus il est possible de spécifier le droit ALL PRIVILEGES qui donne tous les droits à l'utilisateur (sauf celui de transmettre ses droits).
La clause WITH GRANT OPTION est optionnelle, elle permet de préciser que l'utilisateur a le droit de transférer ses propres droits sur la table à d'autres utilisateur. Une telle clause permet une gestion décentralisée de l'attribution des droits et non reposant uniquement dans les mains d'un administrateur unique.
La spécification PUBLIC à la place d'un nom d'utilisateur permet de donner les droits spécifiés à tous les utilisateurs de la BD.

Exemple

```
GRANT SELECT, UPDATE ON Personne TO Pierre;
GRANT ALL PRIVILEGES ON Adresse TO PUBLIC;
```

Remarque : Droits sur un SGBD
Certains SGBD permettent de spécifier des droits au niveau du SGBD, c'est à dire pour toutes les tables de toutes les BD du SGBD. La syntaxe dans le cas de MySQL est "*.*" à la place du nom de la table.
Dans ce cas les droits CREATE et DROP sont généralement ajoutés pour permettre ou non aux utilisateurs de créer et supprimer des BD et des tables.

Remarque : Droits sur une BD
Certains SGBD permettent de spécifier des droits au niveau d'une BD, c'est à dire pour toutes les tables de cette base de données. La syntaxe dans le cas de MySQL est "nom_bd.*" à la place du nom de la table.
Dans ce cas les droits CREATE et DROP sont généralement ajoutés pour permettre ou non aux utilisateurs de créer et supprimer des tables sur cette BD.

Remarque : Droits sur une vue
Il est possible de spécifier des droits sur des vues plutôt que sur des tables, avec une syntaxe identique (et un nom de vue à la place d'un nom de table).

Remarque : Catalogue de données
Les droits sont stockés dans le catalogue des données, il est généralement possible de modifier directement ce catalogue à la place d'utiliser la commande GRANT. Cela reste néanmoins déconseillé.

Remarque : Création des utilisateurs
Les modalités de création d'utilisateurs, voire de groupes d'utilisateurs dans le SGBD, reste dépendantes de celui-ci. Des commande SQL peuvent être disponibles, telles que CREATE USER, ou bien la commande GRANT lorsque qu'elle porte sur un utilisateur non existant peut être chargée de créer cet utilisateur. Des modules spécifiques d'administration sont généralement disponibles pour prendre en charge la gestion des utilisateurs.

2. Révocation de droits

SQL propose une commande pour révoquer les droits attribués à des utilisateurs.

Syntaxe

```
REVOKE <liste de droits> ON <nom table> FROM <utilisateur>
```

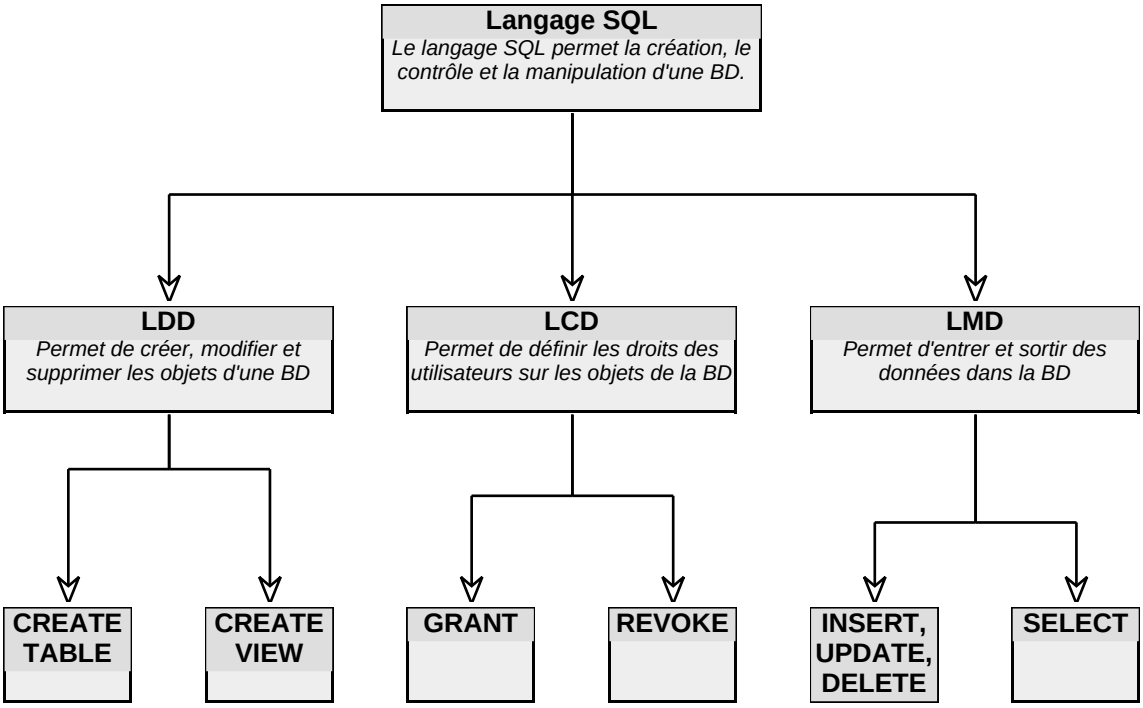
Exemple

```
REVOKE SELECT, UPDATE ON Personne TO Pierre;
REVOKE ALL PRIVILEGES ON Adresse TO PUBLIC;
```

Remarque : Révocation du droit de donner les droits
Pour retirer les droits de donner les droits à un utilisateur (qui l'a donc obtenu par la clause WITH GRANT OPTION), il faut utiliser la valeur GRANT OPTION dans la liste des droits révoqués.

Remarque : Révocation en cascade
Lorsque qu'un droit est supprimé pour un utilisateur, il l'est également pour tous les utilisateurs qui avait obtenu ce même droit par l'utilisateur en question.

En résumé...



Normalisation de relations

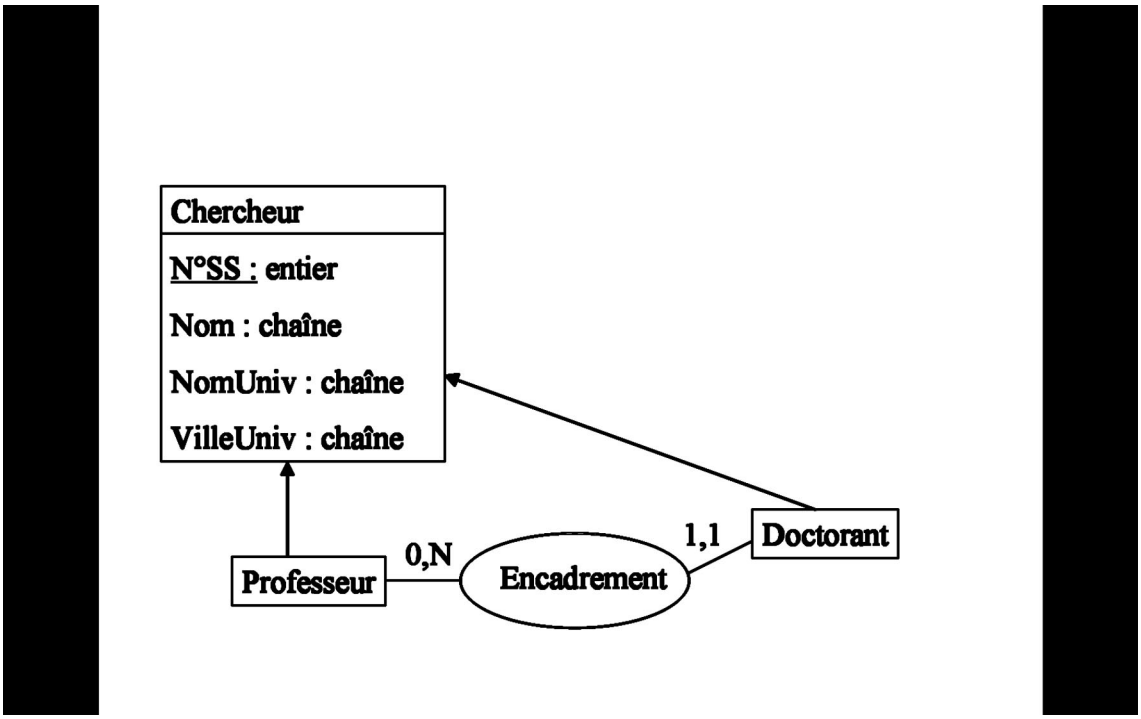
Problème posé

Soit un modèle conceptuel E-A représentant :

- un type d'entité "chercheur", identifié par le numéro de sécurité sociale, et possédant les autres propriétés suivantes : le nom, le nom de l'université à laquelle il appartient, la ville dans laquelle est basée cette université.
- un type d'entité "professeur", héritant de "chercheur"
- un type d'entité "doctorant", héritant de "chercheur"
- une association de type "encadrement" entre professeur et doctorant (un professeur pouvant encadrer plusieurs doctorants et un doctorant n'ayant qu'un et un seul directeur de thèse).

- Dessiner le modèle E-A.
- Traduire le modèle E-A en modèle logique relationnel.
- Après avoir identifié les DF, normaliser le modèle relationnel en BCNF.
- Ecrire les instructions SQL de création d'un tel modèle.

Modèle E-A



▲ IMG. 29

Modèle relationnel

```
Professeur ( N°SS:int(13), Nom:char(20), NomUniv:char(50), VilleUniv:char(20))
Doctorant(N°SS:int(13), Nom:char(20), NomUniv:char(50), VilleUniv:char(20),
EncadrePar=>Professeur)
```

Dépendances Fonctionnelles

Professeur :

- N°SS→ Nom, NomUniv, VilleUniv
- NomUniv→ VilleUniv

Doctorant :

- N°SS→ Nom, NomUniv, VilleUniv, EncadrePar
- NomUniv→ VilleUniv

Formes normales

La schéma n'est pas en 3NF :
NomUniv→ VilleUniv

Normalisation en BCNF

```
Professeur (N°SS:int(13), Nom:char(20), NomUniv=>Univ)
Doctorant (N°SS:int(13), Nom:char(20), NomUniv=>Univ, EncadrePar=>Professeur)
Univ(Nom:char(50) , Ville:char(20))
```

Implémentation SQL

```
Create Table Professeur (
   N°SS INTEGER(13) PRIMARY KEY,
   Nom CHAR(20) NOT NULL,
   NomUniv CHAR(50) REFERENCES Univ(Nom));
```

```
Create Table Doctorant (
   N°SS INTEGER(13) PRIMARY KEY,
   Nom CHAR(20) NOT NULL,
   NomUniv CHAR(50) REFERENCES Univ(Nom),
   EncadrePar INTEGER(13) REFERENCES Professeur(N°SS) );
```

```
Create Table Univ(
   Nom CHAR(50) PRIMARY KEY,
   Ville CHAR(20) );
```

Remarque : Classe chercheur abstraite

On notera que la classe non finale chercheur est toujours considérée comme abstraite en modélisation E-A.

Cours général

V

Transactions

Les transactions sont une réponse générale aux problèmes de fiabilité et d'accès concurrents dans les BD, et en particulier dans les BD en mode client-serveur. Elles sont le fondement de toute implémentation robuste d'une BD. Des systèmes comme Oracle ne fonctionnent nativement qu'en mode transactionnel.

Chapitre A. Gestion des transactions

Objectifs pédagogiques

Comprendre les principes et l'intérêt des transactions
Apréhender la gestion des pannes dans les SGBD
Apréhender la gestion de la concurrence dans les SGBD
Connaître les syntaxes SQL standard, Oracle et Access pour utiliser des transactions
Maîtriser les modalités d'utilisation des transactions

Problématique des pannes et de la concurrence

Une BD est un ensemble persistant de données organisées qui a en charge la préservation de la cohérence de ces données. Les données sont cohérentes si elles respectent l'ensemble des contraintes d'intégrité spécifiées pour ces données : contraintes de domaine, intégrité référentielle, dépendances fonctionnelles, etc.

La cohérence des données peut être remise en cause par deux aspects de la vie d'une BD :

- **La défaillance**

  Lorsque le système tombe en panne alors qu'un traitement est en cours, il y a un risque qu'une partie seulement des instructions prévues soit exécutée, ce qui peut conduire à des incohérences. Par exemple pendant une mise à jour en cascade de clés étrangères suite au changement d'une clé primaire.

- **La concurrence**

  Lorsque deux accès concurrents se font sur les données, il y a un risque que l'un des deux accès rende l'autre incohérent. Par exemple si deux utilisateurs en réseau modifient une donnée au même moment, seule une des deux mises à jour sera effectuée.

La gestion de transactions par un SGBD est à la base des mécanismes qui permettent d'assurer le maintien de la cohérence des BD. C'est à dire encore qu'il assure que tous les contraintes de la BD seront toujours respectées, même en cas de panne et même au cours d'accès concurrents.

Section A1. Transactions

1. Notion de transaction

Transaction
Une transaction est une unité logique de travail, c'est à dire une séquence d'instructions, dont l'exécution assure le passage de la BD d'un état cohérent à un autre état cohérent.

Cohérence des exécutions incorrectes
La transaction assure le maintien de la cohérence des données que son exécution soit *correcte* ou *incorrecte*.

Exemples d'exécutions incorrectes
L'exécution d'une transaction peut être incorrecte parce que :

- Une panne a lieu
- Un accès concurrent pose un problème
- Le programme qui l'exécute en a décidé ainsi

2. Déroulement d'une transaction

1. DEBUT
2. TRAITEMENT
   - Accès aux données en lecture
   - Accès aux données en écriture
3. FIN
   - Correcte : Validation des modifications
   - Incorrecte : Annulation des modifications

Remarque
Tant qu'une transaction n'a pas été terminée correctement, elle doit être assimilée à une *tentative* ou une mise à jour *virtuelle*, elle reste incertaine. Une fois terminée correctement la transaction ne peut plus être annulée par aucun moyen.

3. Propriétés ACID d'une transaction

Une transaction doit respecter quatre propriétés fondamentales :

- **L'atomicité**

  Les transactions constituent l'unité logique de travail, toute la transaction est exécutée ou bien rien du tout, mais jamais une partie seulement de la transaction.

- **La cohérence**

  Les transactions préservent la cohérence de la BD, c'est à dire qu'elle transforme la BD d'un état cohérent à un autre (sans nécessairement que les états intermédiaires internes de la BD au cours de l'exécution de la transaction respectent cette cohérence)

- **L'isolation**

  Les transactions sont isolées les unes des autres, c'est à dire que leur exécution est indépendante des autres transactions en cours. Elles accèdent donc à la BD comme si elles étaient seules à s'exécuter, avec comme corrolaire que les résultats intermédiaires d'une transaction ne sont jamais accessibles aux autres transactions.

- **La durabilité**

  Les transactions assurent que les modifications qu'elles induisent perdurent, même en cas de défaillance du système.

Remarque
Les initiales de Atomicité, Cohérence, Isolation et Durabilité forme le mot mnémotechnique ACID.

4. Transactions en SQL

Le langage SQL fournit trois instructions pour gérer les transactions.

Syntaxe : Début d'une transaction

```
BEGIN TRANSACTION
```

Cette syntaxe est optionnelle (voire inconnue de certains SGBD), une transaction étant débutée de façon *implicite* dès qu'instruction est initiée sur la BD.

Syntaxe : Fin correcte d'une transaction

```
COMMIT TRANSACTION (ou COMMIT)
```

Cette instruction SQL signale la fin d'une transaction couronnée de succès. Elle indique donc au gestionnaire de transaction que l'unité logique de travail s'est terminée dans un état cohérent est que les données peuvent effectivement être modifiées de façon durable.

Syntaxe : Fin incorrecte d'une transaction

```
ROLLBACK TRANSACTION (ou ROLLBACK)
```

Cette instruction SQL signale la fin d'une transaction pour laquelle quelque chose s'est mal passé. Elle indique donc au gestionnaire de transaction que l'unité logique de travail s'est terminée dans un état potentiellement incohérent et donc que les données ne doivent pas être modifiées en annulant les modifications réalisées au cours de la transaction.

Remarque : Programme
Un programme est généralement une séquence de plusieurs transactions.

5. Exemple de transaction sous Oracle

Exemple de gestion de compte toujours positif sur Oracle en PL/SQL

```
DECLARE
   vTotal number;


BEGIN
   UPDATE compte
   SET total=total-1000
   WHERE nom="dupont";


   SELECT total INTO vTotal
   FROM compte
   WHERE nom="dupont";


   IF vTotal<0 THEN
      ROLLBACK;
   ELSE
      COMMIT;
   END IF;
END;
```

6. Exemple de transaction sous Access

Exemple de transfert financier sous Access

```
Sub Transaction
   BeginTrans
   CurrentDb.CreateQueryDef("", "UPDATE Compte1 SET Solde=Solde+100 WHERE
   Num=1").Execute
   CurrentDb.CreateQueryDef("", "UPDATE Compte2 SET Solde=Solde-100 WHERE
   Num=1").Execute
   CommitTrans
End Sub
```

Remarque : VBA et transactions

Sous Access, il n'est possible de définir des transactions sur plusieurs objets requêtes qu'en VBA.

Remarque : Transactions automatique sous Access

Sous Access, toute requête portant sur plusieurs lignes d'une table est (sauf paramétrage contraire) encapsulée dans une transaction.
Ainsi par exemple la requête "UPDATE Compte SET Solde=Solde*6,55957" est exécutée dans une transaction et donc, soit toutes les lignes de la table Compte seront mises à jour, soit aucune.

7. Journal des transactions

Journal

Le journal est un fichier système qui constitue un espace de stockage redondant avec la BD. Il répertorie l'ensemble des mises à jour faites sur la BD (en particulier les valeurs des enregistrements avant et après mise à jour). Le journal est donc un historique *persistant* (donc en mémoire secondaire) qui mémorise tout ce qui se passe sur la BD.

Le journal est indispensable pour la validation (COMMIT), l'annulation (ROLLBACK) et la reprise après panne de transactions.

- *Synonyme : Log.*

Section A2. Fiabilité

1. Les pannes

Une BD est parfois soumise à des défaillances qui entraînent une perturbation, voire un arrêt, de son fonctionnement.

On peut distinguer deux types de défaillances :

- **Les défaillances système**

  ou défaillances douces (*soft crash*), par exemple une coupure de courant ou une panne réseau. Ces défaillances affectent toutes les transactions en cours de traitement, mais pas la BD au sens de son espace de stockage physique.

- **Les défaillances des supports**

  ou défaillances dures (*hard crash*), typiquement le disque dur sur lequel est stockée la BD. Ces défaillances affectent également les transactions en cours (par rupture des accès aux enregistrements de la BD), mais également les données elles-mêmes.

Remarque : Annulation des transactions non terminées
Lorsque le système redémarre après une défaillance, toutes les transactions qui étaient en cours d'exécution (pas de COMMIT) au moment de la panne sont annulés (ROLLBACK imposé par le système). Cette annulation assure le retour à un état cohérent, en vertu des propriétés ACID [Atomique, Cohérent, Isolé, Durable] des transactions.

Remarque : Ré-exécution des transactions terminées avec succès
Au moment de la panne certaines transactions étaient peut-être terminées avec succès (COMMIT) mais non encore (ou seulement partiellement) enregistrées dans la BD (en mémoire volatile, tampon, etc.). Lorsque le système redémarre il doit commencer par rejouer ces transactions, qui assurent un état cohérent de la BD plus avancé.
Cette reprise des transactions après COMMIT est indispensable dans la mesure où c'est bien l'instruction COMMIT qui assure la fin de la transaction et donc la *durabilité*. Sans cette gestion, toute transaction pourrait être remise en cause.

Remarque : Unité de reprise
Les transactions sont des unités de travail, et donc également de reprise.

Remarque : Défaillance des supports
Tandis que la gestion de transactions et de journal permet de gérer les défaillances systèmes, les défaillances des supports ne pourront pas toujours être gérés par ces seuls mécanismes. Il faudra leur adjoindre des procédures de *sauvegarde* et de restauration de la BD pour être capable au pire de revenir dans un état antérieur cohérent et au mieux de réparer complètement la BD (cas de la réplication en temps réel par exemple).

2. Point de contrôle

Point de contrôle
Un point de contrôle est une écriture dans le journal positionnée automatiquement par le système qui établit la liste de toutes les transactions en cours (au moment où le point de contrôle est posé) et force la sauvegarde des données alors en mémoire centrale dans la mémoire secondaire.

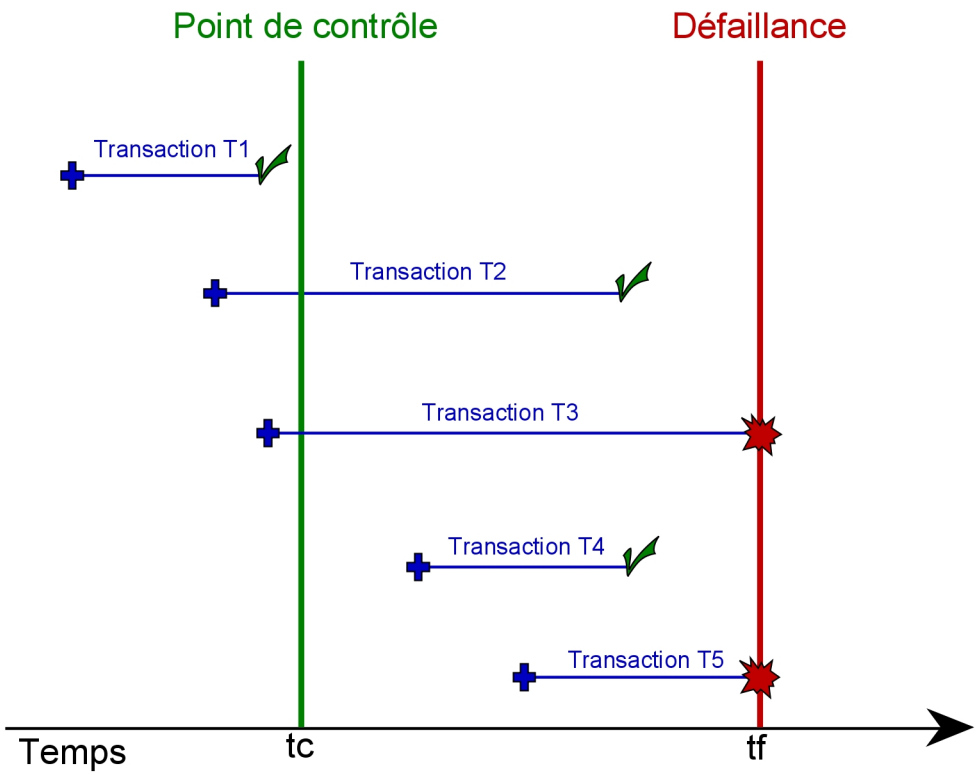
Le point de contrôle est positionné à intervale de temps ou de nombre d'entrées dans le journal prédéfinis.
Le dernier point de contrôle est le point de départ d'une reprise après panne, dans la mesure où c'est le dernier instant où toutes les données ont été sauvegardées en mémoire non volatile.

- *Synonyme : Syncpoint.*

3. Reprise après panne

Le mécanisme de reprise après panne s'appuie sur le journal et en particulier sur l'état des transactions au moment de la panne et sur le dernier point de contrôle.

Le schéma ci-après illustre les cinq cas de figure possibles pour une transaction au moment de la panne.



▲ SCH. 7 : LES CINQ TYPES DE TRANSACTIONS AU MOMENT DE LA PANNE

- **Transactions de type T1**

  Elles ont débuté et se sont terminées avant tc. Elles n'interviennent pas dans le processus de reprise.

- **Transactions de type T2**

  Elles ont débuté avant tc et se sont terminées entre tc et tf. Elles devront être rejouées (il n'est pas sûr que les données qu'elles manipulaient aient été correctement inscrites en mémoire centrale, puisque après tc, or le COMMIT impose la durabilité).

- **Transactions de type T3**

  Elles ont débuté avant tc, mais n'était pas terminées à tf. Elles devront être annulées (pas de COMMIT).

- **Transactions de type T4**

  Elles ont débuté après tc et se sont terminées avant tf. Elles devront être rejouées.

- **Transactions de type T5**

  Elles ont débuté après tc et ne se sont pas terminées. Elles devront être annulées.

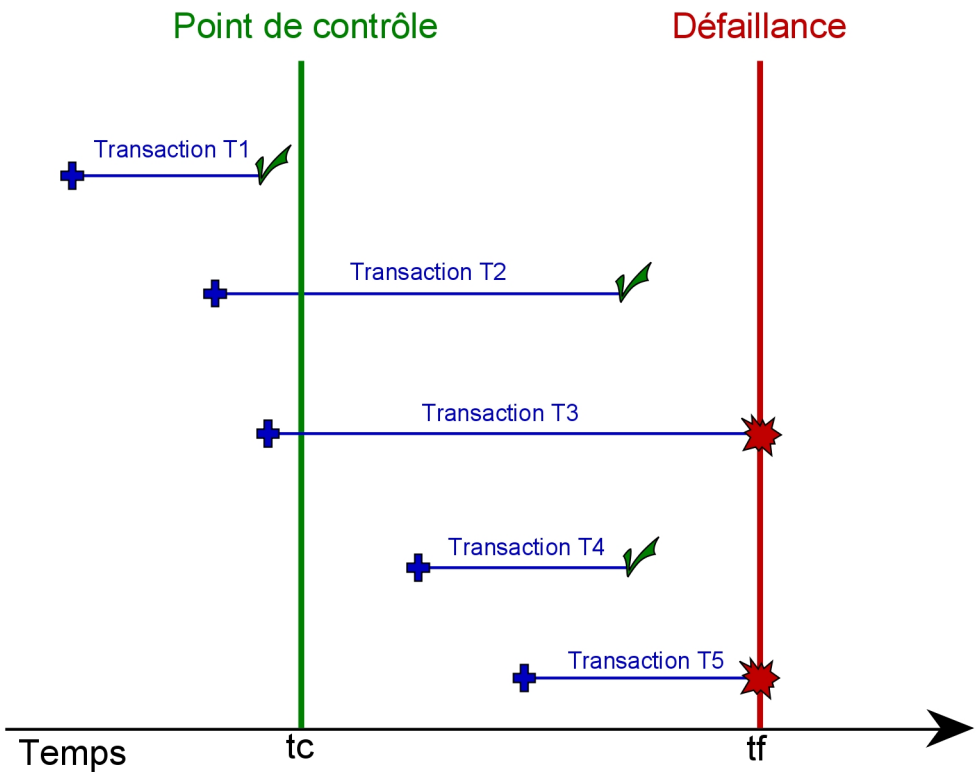
Remarque
Les transactions sont des unités d'intégrité.

4. Algorithme de reprise UNDO-REDO.

L'algorithme suivant permet d'assurer une reprise après panne qui annule et rejoue les transactions adéquates.

```
1. SOIT deux listes REDO et UNDO
   1a. Initialiser la liste REDO à vide
   1b. Initialiser la liste UNDO avec toutes les transactions en cours au dernier
   point de contrôle


2. FAIRE une recherche en avant dans le journal, à partir du point de contrôle
   2a. SI une transaction T est commencée ALORS ajouter T à UNDO
   2b. SI une transaction T est terminée avec succès alors déplacer T de UNDO à
   REDO


3. QUAND la fin du journal est atteinte
   3a. Annuler les transactions de la liste UNDO (reprise en arrière)
   3b. Rejouer les transactions de la liste REDO (reprise en avant)


4. TERMINER la reprise et redevenir disponible pour de nouvelles instructions
```

Exemple



▲ SCH. 8 : LES CINQ TYPES DE TRANSACTIONS AU MOMENT DE LA PANNE

- **Transactions de type T1**

  Non prises en compte par l'algorithme.

- **Transactions de type T2**

  Ajoutées à la liste UNDO (étape 1b) puis déplacée vers REDO (étape 2b) puis rejouée (étape 3b).

- **Transactions de type T3**

  Ajoutées à la liste UNDO (étape 1b) puis annulée (étape 3a).

- **Transactions de type T4**
  Ajoutées à la liste UNDO (étape 2a) puis déplacée vers REDO (étape 2b) puis rejouée (étape 3b).
- **Transactions de type T5**
  Ajoutées à la liste UNDO (étape 2a) puis annulée (étape 3a).

5. Ecriture en avant du journal.

Fondamentaux
On remarquera que pour que la reprise de panne soit en mesure de rejouer les transactions, la première action que doit effectuer le système au moment du COMMIT est l'écriture dans le journal de cette fin correcte de transaction. En effet ainsi la transaction pourra être rejouée, même si elle n'a pas eu le temps de mettre effectivement à jour les données dans la BD, puisque le journal est en mémoire secondaire.

* *
*

On voit que la gestion transactionnelle est un appui important à la reprise sur panne, en ce qu'elle assure des états cohérents qui peuvent être restaurés.

Section A3. Concurrence

1. Trois problèmes soulevés par la concurrence.

Nous proposons ci-dessous trois problèmes posés par les accès conccurents des transactions aux données.

1.1. Perte de mise à jour

Temps	Transaction A	Transaction B
t1	LIRE T	
t2	...	LIRE T
t3	UPDATE T	...
t4	...	UPDATE T
t5	COMMIT	...
t4		COMMIT

▲ TAB. 14 : PROBLÈME DE LA PERTE DE MISE À JOUR DU TUPLE T PAR LA TRANSACTION A

Les transaction A et B accèdent au même tuple T ayant la même valeur respectivement à t1 et t2. Ils modifient chacun la valeur de T. Les modifications effectuées par A seront perdues puisqu'elle avait lu T avant sa modification par B.

Exemple

Temps	A : Ajouter 100	B : Ajouter 10
t1	LIRE COMPTE C=1000	
t2	...	LIRE COMPTE C=1000
t3	UPDATE COMPTE C=C+100=1100	...
t4	...	UPDATE COMPTE C=C+10=1010
t5	COMMIT C=1100	...
t6		COMMIT C=1010

▲ TAB. 15 : DOUCLE CRÉDIT D'UN COMPTE BANCAIRE C

Dans cet exemple le compte bancaire vaut 1010 à la fin des deux transactions à la place de 1110.

1.2. Accès à des données non validées

Temps	Transaction A	Transaction B
t1		UPDATE T
t2	LIRE T	...
t3		ROLLBACK

▲ TAB. 16 : PROBLÈME DE LA LECTURE IMPROPRE DU TUPLE T PAR LA TRANSACTION A

La transaction A accède au tuple T qui a été modifié par la transaction B. B annule sa modification et A a donc accédé à une valeur qui n'aurait jamais dû exister (virtuelle) de T. Pire A pourrait faire une mise à jour de T *après* l'annulation par B, cette mise à jour incluerait la valeur *avant* annulation par B (et donc reviendrait à annuler l'annulation de B).

Exemple

Temps	A : Ajouter 10	B : Ajouter 100 (erreur)
t1		LIRE COMPTE C=1000
t2		UPDATE COMPTE C=C+100=1100
t3	LIRE COMPTE C=1100	...
t4	...	ROLLBACK C=1000
t5	UPDATE C C=C+10=1110	
t6	COMMIT C=1110	

▲ TAB. 17 : ANNULATION DE CRÉDIT SUR LE COMPTE BANCAIRE C

Dans cet exemple le compte bancaire vaut 1110 à la fin des deux transactions à la place de 1010.

1.3. Lecture incohérente

Temps	Transaction A	Transaction B
t1	LIRE T	
t2	...	UPDATE T
t3	...	COMMIT
t4	LIRE T	

▲ TAB. 18 : PROBLÈME DE LA LECTURE NON REPRODUCTIBLE DU TUPLE T PAR LA TRANSACTION A

Si au cours d'une même transaction A accède deux fois à la valeur d'un tuple alors que ce dernier est, entre les deux, modifié par une autre transaction B, alors la lecture de A est inconsistente. Ceci peut entraîner des incohérences par exemple si un calcul est en train d'être fait sur des valeurs par ailleurs en train d'être mises à jour par d'autres transactions.

Remarque
Le problème se pose bien que la transaction B ait été validée, il ne s'agit donc pas du problème d'accès à des données non validées.

Exemple

Temps	A : Calcul de S=C1+C2	B : Transfert de 10 de C1 à C2
t1	LIRE COMPTE1 C1=100	
t2	...	LIRE COMPTE 1 C1=100
t3	...	LIRE COMPTE 2 C2=100
t4	...	UPDATE COMPTE 1 C1=100-10=90
t5	...	UPDATE COMPTE 2 C2=100+10=110
t6	...	COMMIT
t7	LIRE COMPTE 2 C2=110	
t8	CALCUL S S=C1+C2=210	

▲ TAB. 19 : TRANSFERT DU COMPTE C1 AU COMPTE C2 PENDANT UNE OPÉRATION DE CALCUL C1+C2

Dans cet exemple la somme calculée vaut 210 à la fin du calcul alors qu'elle devrait valoir 200.

2. Le verrouillage.

Une solution générale à la gestion de la concurrence est une technique très simple appelée verrouillage.

Verrou
Poser un verrou sur un objet (typiquement un tuple) par une transaction signifie rendre cet objet innaccessible aux autres transactions.

- *Synonyme : Lock.*

Verrou partagé
Un verrou partagé, noté S, est posé par une transaction lors d'un accès en *lecture* sur cet objet.

Un verrou partagé interdit aux autres transaction de poser un verrou exclusif sur cet objet et donc d'y accéder en écriture.

- *Synonymes : Verrou de lecture, Shared lock, Read lock.*

Verrou exclusif
Un verrou exclusif, noté X, est posé par une transaction lors d'un accès en *écriture* sur cet objet.

Un verrou exclusif interdit aux autres transactions de poser tout autre verrou (partagé ou exclusif) sur cet objet et donc d'y accéder (ni en lecture, ni en écriture).

- *Synonymes : Verrou d'écriture, Exclusive lock, Write lock.*

Remarque : Verrous S multiples
Un même objet peut être verrouillé de façon partagée par plusieurs transactions en même temps. Il sera impossible de poser un verrou exclusif sur cet objet tant *qu'au moins une* transaction disposera d'un verrou S sur cet objet.

Règles de verrouillage
Soit la transaction A voulant poser un verrou S sur un objet O
1. Si O n'est pas verrouillé alors A peut poser un verrou S
2. Si O dispose déjà d'un ou plusieurs verrous S alors A peut poser un verrou S
3. Si O dispose déjà d'un verrou X alors A ne peut pas poser de verrou S

Soit la transaction A voulant poser un verrou X sur un objet O
1. Si O n'est pas verrouillé alors A peut poser un verrou X
2. Si O dispose déjà d'un ou plusieurs verrous S ou d'un verrou X alors A ne peut pas poser de verrou X

	Verrou X présent	Verrou(s) S présent(s)	Pas de verrou présent
Verrou X demandé	Demande refusée	Demande refusée	Demande accordée
Verrou S demandé	Demande refusée	Demande accordée	Demande accordée

▲ TAB. 20 : MATRICE DES RÈGLES DE VERROUILLAGE

Remarque : Promotion d'un verrou
Une transaction qui dispose déjà, *elle-même*, d'un verrou S sur un objet peut obtenir un verrou X sur cet objet si aucune autre transaction ne détient de verrou S sur l'objet. Le verrou est alors promu du statut partagé au statut exclusif.

3. Le déverrouillage.

Déverrouillage
Lorsqu'une transaction se termine (COMMIT ou ROLLBACK) elle libère tous les verrous qu'elle a posé.

- *Synonyme : Unlock.*

4. Protocole d'accès aux données.

Règles de verrouillage avant les lectures et écritures des données
Soit la transaction A voulant lire des données d'un tuple T :
1. A demande à poser un verrou S sur T
2. Si A obtient de poser le verrou alors A lit T
3. Sinon A attend le droit de poser son verrou (et donc que les verrous qui l'en empêchent soient levés)

Soit la transaction A voulant écrire des données d'un tuple T :
1. A demande à poser un verrou X sur T
2. Si A obtient de poser le verrou alors A écrit T
3. Sinon A attend le droit de poser son verrou (et donc que les verrous qui l'en empêchent soient levés)

Soit la transaction A se terminant (COMMIT ou ROLLBACK) :
1. A libère tous les verrous qu'elle avait posé
2. Certaines transactions en attente obtiennent éventuellement le droit de poser des verrous

Remarque : Liste d'attente
Afin de rationnaliser les attentes des transactions, des stratégies du type FIFO [First In First Out] sont généralement appliquées et donc les transactions sont empilées selon leur ordre de demande.

5. Solution aux trois problèmes soulevés par la concurrence.

Le principe du verrouillage permet d'apporter une solution aux trois problèmes classiques soulevés par les accès aux concurrents aux données par les transactions.

5.1. Perte de mise à jour

Temps	Transaction A	Transaction B
t1	LIRE T Verrou S	
t2	...	LIRE T Verrou S
t3	UPDATE T Attente...	...
t4	... Attente...	UPDATE T Attente...
...	Inter-blocage	

▲ TAB. 21 : PROBLÈME DE LA PERTE DE MISE À JOUR DU TUPLE T PAR LA TRANSACTION A

Remarque
Le problème de perte de mise à jour est réglé, mais soulève ici un autre problème, celui de *l'inter-blocage*.

5.2. Accès à des données non validées

Temps	Transaction A	Transaction B
t1		UPDATE T Verrou X
t2	LIRE T Attente...	...
t3		ROLLBACK Libération du verrou X
t4	Verrou S	

▲ TAB. 22 : PROBLÈME DE LA LECTURE IMPROPRE DU TUPLE T PAR LA TRANSACTION A

5.3. Lecture incohérente

Temps	Transaction A	Transaction B
t1	LIRE T Verrou S	
t2	...	UPDATE T Attente...
t3	LIRE T Verrous S	...
...	... libération des verrous ...	... reprise de la transaction ...

▲ TAB. 23 : PROBLÈME DE LA LECTURE NON REPRODUCTIBLE DU TUPLE T PAR LA TRANSACTION A

Remarque
La lecture reste cohérente car aucune mise à jour ne peut intervenir pendant le processus de lecture d'une même transaction.

6. Inter-blocage

Inter-blocage
L'inter-blocage est le phénomène qui apparait quand deux transactions (ou plus, mais généralement deux) se bloquent mutuellement par des verrous posés sur les données. Ces verrous empêchent chacune des transactions de se terminer et donc de libérer les verrous qui bloquent l'autre transaction. Un processus d'attente sans fin s'enclenche alors.

Les situations d'inter-blocage sont détectées par les SGBD et gérées, en annulant l'une, l'autre ou les deux transactions, par un ROLLBACK système. Les méthodes utilisées sont la détection de cyle dans un graphe d'attente et la détection de délai d'attente trop long.

- *Synonymes : Deadlock, Blocage, Verrou mortel.*

Cycle dans un graphe d'attente
Principe de détection d'un inter-blocage par détection d'un cycle dans un graphe représentant quelles transactions sont en attente de quelles transactions (par inférence sur les verrous posés et les verrous causes d'attente). Un cycle est l'expression du fait qu'une transaction A est en attente d'une transaction B qui est en attente d'une transaction A.

La détection d'un tel cycle permet de choisir une *victime*, c'est à dire une des deux transactions qui sera annulée pour que l'autre puisse se terminer.

- *Synonyme : Circuit de blocage.*

Délai d'attente
Principe de décision qu'une transaction doit être abandonnée (ROLLBACK) lorsque son délai d'attente est trop long.

Ce principe permet d'éviter les situations d'inter-blocage, en annulant une des deux transactions en cause, et en permettant donc à l'autre de se terminer.

- *Synonyme : Timeout.*

Remarque : Risque lié à l'annulation sur délai
Si le délai est trop court, il y a un risque d'annuler des transactions en situation d'attente longue, mais non bloquées.
Si le délai est trop long, il y a un risque de chute des performances en situation d'inter-blocage (le temps que le système réagisse).
La détection de cycle est plus adaptée dans tous les cas, mais plus complexe à mettre en oeuvre.

Remarque : Relance automatique
Une transaction ayant été annulée suite à un inter-blocage (détection de cycle ou de délai d'attente) n'a pas commis de "faute" justifiant son annulation. Cette dernière est juste dûe aux contraintes de la gestion de la concurrence. Aussi elle n'aurait pas dû être annulée et devra donc être éxécutée à nouveau.
Certains SGBD se charge de relancer automatiquement les transactions ainsi annulées.

Section A4. Illustrations sous Oracle

1. Validation et annulation de transaction

```
CREATE TABLE TransTest (X number(1));
Table créée.


INSERT INTO TransTest VALUES (1);
1 ligne créée.


COMMIT;
Validation effectuée.


INSERT INTO TransTest VALUES (2);
1 ligne créée.


ROLLBACK;
Annulation (rollback) effectuée.


SELECT * FROM TransTest;
X
-----
1
```

2. Validation conditionnelle de transaction

```
CREATE TABLE TransTest (X number(1));
INSERT INTO TransTest VALUES (1);
INSERT INTO TransTest VALUES (1);
COMMIT;


SELECT * FROM TransTest;


DECLARE
   minX TransTest.X%TYPE;
BEGIN
   UPDATE TransTest SET X=X-1;
   SELECT min(X) INTO minX FROM TransTest;
   IF minX < 0 THEN
      ROLLBACK;
   ELSE
      COMMIT;
   END IF;
END;


SELECT * FROM TransTest;
```

Première exécution

```
X
---
1
2


Procédure PL/SQL terminée avec succès.


X
---
0
1
```

Seconde exécution

```
X
---
0
1


Procédure PL/SQL terminée avec succès.


X
---
0
1
```

3. Simulation de panne

```
CREATE TABLE TransTest (X number(1));
INSERT INTO TransTest VALUES (1);
COMMIT;


INSERT INTO TransTest VALUES (2);
SELECT * FROM TransTest;


DECLARE
   i number(5);
BEGIN
   i:=0;
   WHILE i=0 LOOP
      i:=0;
   END LOOP;
END;


COMMIT;
```

Exécution

```
Table créée.
1 ligne créée.
Validation effectuée.


1 ligne créée.
X
---
1
2


... Attente dans la boucle infinie ...
Simulation de panne (en brisant la connection)
```

```
SELECT * FROM TransTest;
X
---
1
```

4. Mises à jour concurrentes

Préparation

```
CREATE TABLE TransTest (X number(1));
INSERT INTO TransTest VALUES (1);
COMMIT;
```

Début exécution 1

```
UPDATE TransTest SET X=X+1 WHERE X=1;
1 ligne mise à jour.


SELECT * FROM TransTest;
X
---
2
```

Début exécution 2

```
SELECT * FROM TransTest;
X
---
1


UPDATE TransTest SET X=X+1 WHERE X=1;
-- ... Mise en attente ...
```

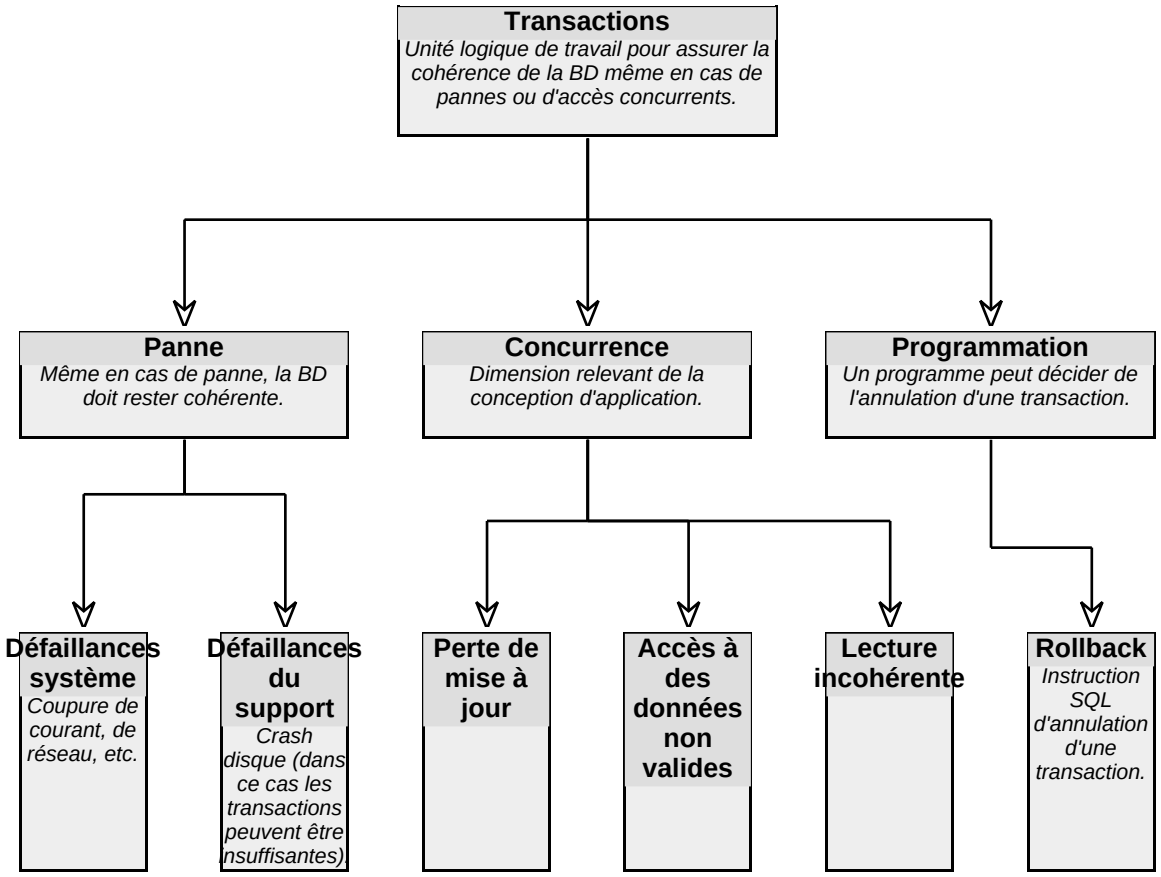
Fin exécution 1

```
COMMIT;
Validation effectuée.
```

Fin exécution 2

```
-- ... Reprise et exécution du UPDATE.
0 ligne(s) mise(s) à jour.
-- NB : X ne vaut plus 1, mais 2 après le COMMIT de la première exécution


COMMIT;
Validation effectuée.


SELECT * FROM TransTest;
X
---
2
```

En résumé...



Pour aller plus loin...

"http://www.developpez.com/hcesbronlavau/Transactions.htm", Les transactions, **CESBRON LAVAU H,** janvier, 2004.

Une bonne introduction courte au principe des transactions avec un exemple très bien choisi.

Des exemples d'implémentation sous divers SGBD (InterBase par exemple)

"http://eric.univ-lyon2.fr/~jdarmont/docs/sise-bd-td2.pdf", TD2 : Oracle SQL, **DARMONT J,** janvier, 2004.

Des suggestion de tests pour l'expérimentation de transactions sous oracle.

"http://www-inf.int-evry.fr/COURS/BD/BD_REL/SUPPORT/poly.html#RTFToC30", Contrôle des accès concurrents et reprise, **DEFUDE B,** janvier, 2004.

Un aperçu général de l'ensemble de la problématique des transactions, de la concurrence et de la fiabilité.

DELMAL P, "SQL2 SQL3, applications à Oracle", De Boeck Université, 2001.

Une bonne description des principes des transactions, avec les exemples caractéristiques, l'implémentation SQL et une étude de cas sous Oracle 8 (chapitre 5).

MATA-TOLEDO R, CUSHMAN P, "Programmation SQL", Ediscience, 2003.

Un exemple d'exécution de transactions (pages 20-23)

Cours général

VI

SGBDRO

Si le modèle logique relationnel a prouvé sa puissance et sa fiabilité au cours des 20 dernières années, les nouveaux besoins de l'informatique industrielle ont vu l'émergence de structures de données complexes mal adaptées à une gestion relationnelle. La naissance du courant "orienté objet" et des langages associées (Java et C++ par exemple) ont donc également investi le champ des SGBD afin de proposer des solutions pour étendre les concepts du relationnel et ainsi mieux répondre aux nouveaux besoins de modélisation.

Chapitre A. Relationnel-objet

Section A1. Introduction : R, OO, RO

Objectifs pédagogiques

Comprendre les limites du modèle relationnel
Comprendre pourquoi et comment le modèle relationnel peut être étendu

1. Les atouts du modèle relationnel

- Fondé sur une théorie rigoureuse et des principes simples
- Mature, fiable, performant
- Indépendance programme et données
- SGBDR : les plus utilisés, connus, maîtrisés
- SQL une implémentation standard du modèle relationnel, avec des API [Application Program Interface] pour la plupart des langages de programmation
- Les SGBDR incluent des outils performants de gestion de requêtes, de générateurs d'applications, d'administration, d'optimisation, etc, ...

2. Les inconvénients du modèle relationnel

- La structure de donnée en tables est pauvre d'un point de vue de la modélisation logique
- Le *mapping* MCD vers MLD entraîne une perte de sémantique
- La manipulation de structures relationnelles par des langages objets entraîne une *impedance mismatch*, c'est à dire un décalage entre les structures de données pour le stockage et les structures de données pour le traitement (ce qui implique des conversions constantes d'un format à l'autre)
- La 1NF est innapropriée à la modélisation d'objets complexes
- La normalisation entraîne la genèse de structures de données complexes et très fragmentées, qui peuvent notamment poser des problèmes de performance ou d'évolutivité
- Le SQL doit toujours être combiné à d'autres langages de programmation pour être effectivement mis en oeuvre
- La notion de méthode ne peut être intégrée au modèle logique, elle doit être gérée au niveau de l'implémentation physique
- Les types de données disponibles sont limités et non extensibles

3. Les SGBDOO

Les SGBDOO ont été créés pour gérer des structures de données complexes, en profitant de la puissance de modélisation des modéles objets et de la puissance de stockage des BD classiques.

Objectifs des SGBDOO :
- Offrir aux langages de programmation orientés objets des modalités de stockage permanent et de partage entre plusieurs utilisateurs
- Offrir aux BD des types de données complexes et extensibles
- Permettre la représentation de structures complexes et/ou à taille variable

Avantages des SGBDOO :
- Le schéma d'une BD objet est plus facile à appréhender que celui d'une BD relationnelle (il contient plus de sémantique, il est plus proche des entités réelles)
- L'héritage permet de mieux structurer le schéma et de factoriser certains éléments de modélisation
- La création de ses propres types et l'intégration de méthodes permets une représentation plus directe du domaine
- L'identification des objets permet de supprimer les clés artificielles souvent introduites pour atteindre la 3NF et donc de simplifier le schéma
- Les principes d'encapsulation et d'abstraction du modèle objet permettent de mieux séparer les BD de leurs applications (notion d'interface).

Inconvénient des SGBDOO :
- Gestion de la persistance et de la coexistence des objets en mémoire (pour leur manipulation applicative) et sur disque (pour leur persistance) complexe
- Gestion de la concurrence (transactions) plus difficile à mettre en oeuvre
- Interdépendance forte des objets entre eux
- Gestion des pannes
- Complexité des systèmes (problème de fiabilité)
- Problème de compatibilité avec les SGBDR classiques

Explication
Les SGBDOO apportent donc des concepts fondamentaux pour l'évolution des BD, mais leur réalité est encore en grande partie du domaine de la recherche ou d'applications "de niche".
Ils apportent donc une innovation sur des aspects que les SGBDR ne savent pas faire, mais sans être au même niveau sur ce que les SGBDR savent bien faire.

4. Les SGBDRO

Les SGBDRO sont nés du double constat de la puissance nouvelle promise par les SGBDOO et de l'insuffisance de leur réalité pour répondre aux exigences de l'industrie des BD classiques. Leur approche est plutôt d'introduire dans les SGBDR les concepts apportés par les SGBDOO plutôt que de concevoir de nouveaux systèmes.

Objectifs additionnels des SGBDRO :
- Gérer des données complexes (temps, géo-référencement, multimédia, types utilisateurs, etc.)
- Rapprocher le modèle logique du modèle conceptuel
- Réduire l'Impedance mismatch

Section A2. Le modèle relationnel-objet

Objectifs pédagogiques
Connaître les caractéristiques principales du modèle relationnel-objet
Comprendre les avantages du modèle relationnel-objet

1. Les SGBDRO

Modèle relationnel-objet
Modèle relationnel étendu avec des principes objet pour en augmenter les potentialités.

- *Synonyme : Modèle objet-relationnel.*

Les apports principaux du modèle relationnel-objet sont :
- Les types utilisateurs et l'encapsulation (recours aux méthodes)
- La gestion de collections
- L'héritage et la réutilisation
- L'identité d'objet et les références physiques

2. Le modèle imbriqué

- *nested model*
- La 1NF est relachée pour permettre d'affecter des valeurs non atomiques à un attribut, pour modéliser des objets complexes.
- Gestion directe des attributs multivalués, sans passer par une nouvelle relation
- Gestion directe des attributs composés
- Un attribut ne sera plus seulement valuée par une valeur simple, mais pourra l'être par une collection d'objets complexes

Exemple

Personnes				
Nom	Prénom	Age	Voiture	
			Marque	Modèle

▲ IMG. 30 : EXEMPLE DE TABLE IMBRIQUÉE

3. Les types utilisateurs

Type de données utilisateur
Type de données créé par le concepteur d'un schéma relationnel-objet, qui encapsule des données et des opérations sur ces données. Ce concept est à rapprocher de celui de classe d'objets.

- *Synonymes : Type utilisateur, Type de données abstrait.*

Syntaxe

```
Type nom_type : <
   attribut1 typeattribut1,
   attribut2 typeattribut2,
   ...
   attributN typeattributN,
   =methode1 (paramètres) typeretourné1,
   =methode2 (paramètres) typeretourné2,
   =...
   =methodeN (paramètres) typeretournéN
>
```

Remarque
Les type des attributs ou des méthodes d'un type peuvent également être des types utilisateurs.

Exemple : Adresse en relationnel

```
employe (nom:chaîne, prenom:chaîne, ad_num:chaîne, ad_rue:chaîne,
ad_ville:chaîne, ad_cp:chaîne, ad_ville:chaîne, ad_pays:chaîne,
societe=>societe)


societe (nom:chaîne, ad_num:chaîne, ad_rue:chaîne, ad_ville:chaîne,
ad_cp:chaîne, ad_ville:chaîne, ad_pays:chaîne)
```

Exemple : Adresse en relationnel-objet avec type

```
Type adresse : <num:chaîne, rue:chaîne, ville:chaîne, cp:chaîne,
ville:chaîne, pays:chaîne>


employe (nom:chaîne, prenom:chaîne, ad:adresse, societe=>societe)


societe (nom:chaîne, ad:adresse)
```

Exemple : Type adresse avec méthode

```
Type adresse : <
num:chaîne,
   rue:chaîne,
   ville:chaîne,
   cp:chaîne,
   ville:chaîne,
   pays:chaîne,
   =CPprefixé() chaîne
>
```

Remarque : Première forme normale
Le recours aux types utilisateurs brise une règle de la première forme normale, celle de l'atomicité des valeurs d'attribut.
Ce non respect de la 1NF ne pose pas de problème car la déclaration de type permet d'accéder à la

structure interne des attributs composés.
Dans le modèle relationnel classique, le problème du non respect de la 1NF était bien l'opacité de la structure interne des attributs ainsi constitués qui interdisait l'accès à des sous informations extraites de la valeur de l'attribut (par exemple un attribut comprenant le nom et le prénom ensemble interdit l'accès à l'information "nom" ou à l'information "prénom" indépendamment). L'usage de type éclaire cette opacité et rend ainsi le respect de l'atomicité superflu.

4. Les collections

Collection
Une collection est un type de données générique défini afin de supporter les attributs multi-valués.

- *Synonyme : Collection d'objets.*

Syntaxe

```
Type nom_type : collection de <type_objet>
```

Remarque
Les objets d'une collection peuvent être d'un type utilisateur.

Exemple : Collection d'entiers

```
Type liste_telephone : collection de <entier>
```

Exemple : Collection d'objets complexe

```
Type adresse : <num, rue, ville>
Type liste_adresse : collection de <adresse>
```

5. Comparaison relationnel et relationnel-objet

Exemple : Documents en relationnel

```
document (ISBN:entier, titre:chaîne, soustitre:chaîne, editeur:chaîne,
annee:entier)

auteur (id:entier, nom:chaîne, prenom:chaîne)

motscles (document=>document, motcle=>motcle

auteurslivres(idauteur=>auteur, ISBN=>document
```

Exemple : Documents en relationnel-objet

```
Type titre : <titre, soustitre>
Type edition : <titre, soustitre>
Type liste_motscles : collection de <chaîne>
Type liste_auteurs : collection de <auteur>
Type auteur : <nom, prenom>

document (ISBN, titre:titre, edition:edition, motscles:liste_motscles,
auteurs:liste_auteurs)
```

Remarque : Perte de sémantique
La transformation précédente du modèle initiale, a éliminé l'entité auteur. La notion d'auteur n'est donc plus qu'une propriété des livres et non un objet défini indépendamment dans le schéma de donnée. Dans le cas général cette modélisation ne sera pas souhaitable et on préfèrera conserver l'entité auteur. Bien entendu l'existance préalable d'un schéma conceptuel permet de lever l'ambiguïté puisque selon que les auteurs seront définis commes des entités ou bien comme des propriétés de l'entité livre, le choix de modélisation sera effectué.

Exemple : Documents et auteurs en relationnel-objet

```
Type titre : <titre, soustitre>
Type edition : <titre, soustitre>
Type liste_motscles : collection de <chaîne>
Type liste_auteurs : collection de <number>


document (ISBN, titre:titre, edition:edition, motscles:liste_motscles,
auteurs:liste_auteurs)


auteur (id:entier, nom:chaîne, prenom:chaîne)
```

Remarque : Association N:M imbriquée

Dans l'exemple ci-dessus, le schéma relationnel-objet a permis de se débarrasser de la table "auteurslivres" du modèle initial et qui n'existait que pour modéliser l'association N:M entre les livres et les auteurs. Dans l'exemple ci-dessus l'utilisation d'une collection a permis de simplifier le schéma en le débarassant de cette relation artificielle. Notons néanmoins que le fait d'avoir choisi d'imbriquer les auteurs dans les livres (et non l'inverse qui était également possible) est porteur de sens tant au niveau conceptuel (ce sont plutôt les livres qui nous intéressent que les auteurs), qu'au niveau opérationnel (il sera plus facile et plus efficace de trouver les auteurs d'un livre que les livres d'un auteur).

6. Tables d'objets

Une table peut être définie en référençant un type de données plutôt que par des instructions LDD classiques.

On parle alors de table d'objets.

Syntaxe

```
nom_table de nom_type (attributs_clés)
```

Remarque : OID

Une telle définition de table peut permettre d'identifier les objets par un OID [Object Identifier]

Remarque : Héritage

Cette modalité de définition de schéma permet de profiter de l'héritage de type pour permettre l'héritage de schéma de table.

7. Héritage et réutilisation de types

Héritage de type

Un type de données utilisateur peut hériter d'un autre type de données utilisateur.

Syntaxe

```
Type sous_type hérite de type : <
   attributs et méthodes spécifiques
>
```

Remarque : Héritage de schéma de table

Pour qu'une table hérite du schéma d'une autre table, il faut définir les tables depuis des types.

L'héritage entre les types permet ainsi l'héritage entre les schémas de table.

8. Identification d'objets et références

Le modèle relationnel-objet permet de disposer d'identificateurs d'objet (OID)

Caractéristiques

- L'OID est une référence unique pour toute la base de données qui permet de référencer un enregistrement dans une table d'objets.
- L'OID est une référence physique (adresse disque) construit à partir du stockage physique de l'enregistrement dans la base de données.

Avantages

- Permet de créer des associations entre des objets sans effectuer de jointure (gain de performance).
- Fournit une identité à l'objet indépendamment de ses valeurs (clé artificielle).
- Fournit un index de performance maximale (un seul accès disque).
- Permet de garder en mémoire des identificateurs uniques dans le cadre d'une application objet, et ainsi gérer la persistance d'objets que l'on ne peut pas garder en mémoire, avec de bonnes performances (alors que sinon il faudrait exécuter des requêtes SQL pour retrouver l'objet)[Voir exemple "Cadre d'usage" ci-après].

Inconvénient

- Plus de séparation entre le niveau logique et physique.
- L'adresse physique peut changer si le schéma de la table change (changement de la taille d'un champs par exemple)
- Manipulation des données plus complexes, il faut un langage applicatif au dessus de SQL pour obtenir, stocker et gérer des OID dans des variables.

Exemple : Cadre d'usage
L'usage d'OID est pertinent dans le cadre d'applications écrites dans un langage objet, manipulant un très grand nombre d'objets reliés entre eux par un réseau d'associations complexe.
En effet si le nombre d'objets est trop grand, les objets ne peuvent tous être conservés en mémoire vive par l'application objet qui les manipule. Il est alors indispensable de les faire descendre et remonter en mémoire régulièrement. Or dans le cadre d'un traitement portant sur de très nombreux objets, la remonté en mémoire d'un objet est un point critique en terme de performance. A fortiori si l'identification de l'objet à remonter demande une interrogation complexe de la BD, à travers de nombreuses jointures. Le fait d'avoir conservé en mémoire un OID permet de retrouver et de recharger très rapidement un objet, sans avoir à le rechercher à travers des requêtes SQL comportant des jointures et donc très couteuses en performance.

Remarque : Débat
la communauté des BD est plutôt contre les OID, qui rompent avec la séparation entre manipulation logique et stockage physique des données.
La communauté objet est à l'origine de l'intégration des OID dans SQL3, en tant qu'ils sont une réponse au problème de persistence dans les applications objets.

Syntaxe : Référence en utilisant l'OID
Dans une définition de table ou de type :

```
attribut REF nom_table_d_objets
```

Section A3. Mapping conceptuel vers relationnel-objet

Objectifs pédagogiques

Savoir déduire un modèle logique relationnel-objet depuis un modèle conceptuel E-A ou UML.
Intégrer les apports du modèle relationnel-objet par rapport au relationnel dans ce processus

Cette partie permet de traiter la traduction d'un modèle conceptuel UML ou E-A en modèle relationnel-objet. Le modèle E-A étendu est bien plus adapté au relationnel-objet que le modèle E-A classique.

1. Classe

Pour chaque classe (ou entité), créer un type d'objet avec les attributs de la classe (ou entité).

Créer une table d'objets de ce type pour instancier la classe (ou entité), en ajoutant les contraintes d'intégrité.

Remarque : Table d'objet
La fait d'instancier la classe (l'entité) via une table d'objets plutôt qu'une relation classique, permet l'accès à l'héritage de type, à l'implémentation des méthodes et éventuellement à l'usage d'OID à la place de clés étrangères classiques.
Si aucun de ces aspects n'est utilisé, il est possible d'instancier directement la classe (l'entité) en table. Mais le passage par un type est a priori plus systématique.

Remarque : Mapping des méthodes
Si le modèle conceptuel autorise l'expression de méthodes (UML ou extension de E-A), alors on ajoute la définition de ces méthodes sur le type d'objet créé pour de chaque classe (ou entité).

2. Attributs composites

Pour chaque type d'attribut composé, créer un type d'objet.

3. Attributs multi-valués

Pour chaque attribut multi-valué créer une collection du type de cet attribut.

Remarque : Attributs composites multi-valués
Si l'attribut multi-valué est composite, alors créer une collection d'objets.

4. Attributs dérivés

Pour chaque attribut dérivé créer une méthode associée au type d'objet de la classe (l'entité).

5. Association 1:N

Les associations 1:N sont gérées comme en relationnel.

On peut néanmois favoriser l'usage d'objets pour les clés étrangères composées de plusieurs attributs, ainsi que pour les attributs migrants de l'association vers le relation côté N.
Il est aussi possible de gérer la clé étrangère avec un OID à la place d'une clé étrangère classique.

6. Association N:M

Les associations N:M peuvent être gérées comme en relationnel.

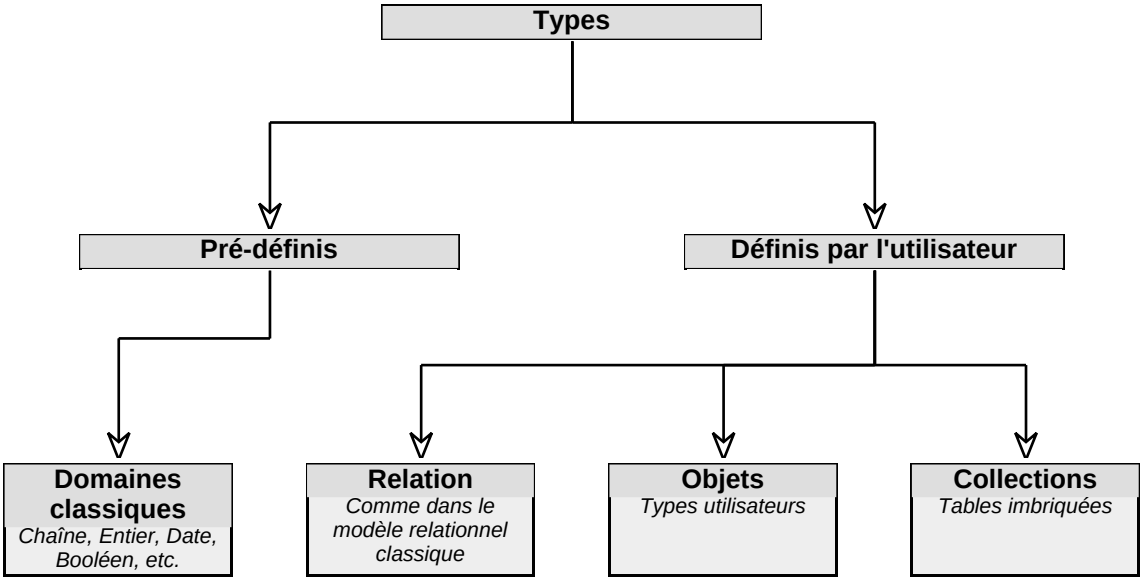
Il est aussi possible de gérer ces relations, en utilisant une collection de références (une collection de clés étrangères) (NB : on favorise ainsi une des deux relations).
Il est aussi possible de gérer ces relations en utilisant un OID comme clé étrangère.
Il est aussi possible de gérer ces relations en utilisant une collection de référence OID.

Remarque : Collection de clés étrangères
Oracle n'autorise pas (à ma connaissance ...) la spécification de contraintes d'intégrité référentielle dans une table imbriquée. Ce qui limite l'utilisation de telles tables pour gérer des relations N:M avec des collections de clés étrangères.
Par contre cela fonctionne avec des références à des OID.

7. Héritage

L'héritage est géré comme en relationnel classique, avec la possibilité de profiter de l'héritage de type pour éliminer la redondance dans la définition des schémas.

En résumé...



Pour s'exercer...

"http://eric.univ-lyon2.fr/~jdarmont/docs/sise-bd-td1isea.pdf", Relationnel-objet, Relationnel étendu, **DARMONT J,** janvier, 2004.

Des exercices corrigés sur le relationnel-objet.

Chapitre B. Implémentation du RO

Section B1. SQL3 (implémentation Oracle 9i)

Objectifs pédagogiques

Connaître l'extension du LDD, et en particulier les types abstrait, les relations imbriquées et l'héritage.
Connaître l'exension du MDL pour naviguer dans les objets manipulés

1. Les nouveaux types de données

SQL3 introduit de nouveaux types de données :
- Large Object (CLOB and BLOB)
- Boolean
- Array
- Row

2. Les types de données abstraits

Implémentation des types utilisateurs.

Syntaxe : Déclaration de type de données

```
CREATE TYPE nom_type AS OBJECT (
   nom_attribut1 type_attribut1
   ...
   MEMBER FUNCTION nom_fonction1 (parametre1 IN|OUT type_parametre1, ...) RETURN
   type_fonction1
   ...
) [NOT FINAL];


CREATE TYPE BODY nom_type
   IS
   MEMBER FUNCTION nom_fonction1 (...) RETURN type_fonction1
      IS
      BEGIN
      ...
      END
   MEMBER FUNCTION nom_fonction2 ...
      ...
   END
```

Syntaxe : Héritage de type

```
CREATE TYPE sous_type UNDER sur_type (
   Déclarations spécifiques ou surcharges
)
```

Remarque : NOT FINAL
Pour être "héritable", un type doit être déclaré avec la clause optionnelle NOT FINAL.

3. Méthodes et SELF

SELF
Lorsque l'on écrit une méthode on a généralement besoin d'utiliser les attributs propres (voire d'ailleurs les autres méthode), de l'objet particulier que l'on est en train de manipuler.
On utilise pour cela la syntaxe SELF qui permet de faire référence à l'objet en cours.

Syntaxe : SELF

```
self.nom_attribut
self.nom_méthode(...)
```

Exemple : Total d'une facture

```
MEMBER FUNCTION total RETURN number
IS
   t number;
BEGIN
   SELECT sum(f.qte) INTO t
   FROM facture f
   WHERE f.num=self.num;
   RETURN t;
END total;
```

Remarque : SELF implicite
Dans certains cas simples, lorsqu'il n'y a aucune confusion possible, SELF peut être ignoré et le nom de l'attribut ou de la méthode directement utilisé.
Il est toutefois plus systématique et plus clair de mettre explicitement le self.

Exemple de SELF implicite

```
MEMBER FUNCTION adresse RETURN varchar2
IS
BEGIN
   RETURN num || rue || ville;
END;
```

4. Nested tables

Implémentation des collections sous forme de tables imbriquées.

Syntaxe : Déclaration de type de données

```
--Création d'un type abstrait d'objet
CREATE TYPE nom_type AS OBJECT (...);


-- Déclaration d'un type abstrait de collection
CREATE TYPE liste_nom_type AS TABLE OF nom_type;


-- Déclaration d'une table imbriquant une autre table
CREATE TABLE nom_table_principale (
   nom_attribut type_attribut,
   ...
   nom_attribut_table_imbriquée liste_nom_type,
   ...
   )
   NESTED TABLE nom_attribut_table_imbriquée STORE AS nom_table_stockage;
```

5. Tables d'objets

Implémentation d'une définition de table depuis un type.

Syntaxe : Déclaration de type de données

```
CREATE TABLE nom_table_principale OF nom_type (
   PRIMARY KEY(attribut1),
   attribut2 NOT NULL,
   UNIQUE (attribut3)
   FOREIGN KEY (attribut4) REFERENCES ...
);
```

Remarque : Méthodes
Si le type sur lequel s'appuie la création de la table définit des méthodes, alors les méthodes seront associées à la table.

Remarque : Contraintes
Il est possible, sur une table ainsi définie, de spécifier les mêmes contraintes que pour une table créée avec une clause CREATE TABLE (contraintes de table). Ces contraintes doivent être spécifiées au moment de la création de la table, et non au moment de la création du type. (bien que la définition d'objet permet de spécifier des contraintes du type NOT NULL, etc.)

6. Insertion d'objets

La manipulation de données est étendue pour assurer la gestion des objets et des collections.

Remarque : Constructeur d'objet
Pour tout type d'objet est automatiquement créée une méthode de construction, dont le nom est celui du type et les arguments les attributs du type.

Syntaxe : Insertion d'objets

```
INSERT INTO nom_table (..., attribut_type_objet, ...)
VALUES
(..., nom_type(valeur1, valeur2, ...), ...);
```

Syntaxe : Insertion de collections d'objets

```
INSERT INTO nom_table (..., attribut_type_collection, ...)
VALUES (
   ...
   nom_type_collection(
      nom_type_objet(valeur1, valeur2, ...),
      nom_type_objet(valeur1, valeur2, ...),
      ...);
   ...
);
```

7. Sélection dans des objets

Syntaxe : Accès aux attributs des objets

```
SELECT alias_table.nom_objet.nom_attribut
FROM nom_table AS alias_table
```

Syntaxe : Accès aux méthodes des objets

```
SELECT alias_table.nom_objet.nom_méthode(paramètres)
FROM nom_table AS alias_table
```

Remarque : Alias obligatoire
L'accès aux objets se fait obligatoirement en préfixant le chemin d'accès par l'alias de la table et non directement par le nom de la table.

8. Sélection dans des tables imbriquées

Syntaxe : Accès aux tables imbriquées
Soit "nt" une table imbriquée dans la table "t".

```
SELECT t2.*
FROM t t1, TABLE(t1.nt) t2;
```

Remarque : Jointure avec la table imbriquée
La jointure entre la table principale et sa table imbriquée est implicitement réalisée, il n'est pas besoin de la spécifier.

Remarque : Accès à la colonne d'une table imbriquée de scalaires
Lorsque la table imbriquée est une table de type scalaire (et non une table d'objets d'un type utilisateur), alors la colonne de cette table n'a pas de nom (puisque le type est scalaire la table n'a qu'une colonne). Pour accéder à cette colonne, il faut utiliser la syntaxe dédiée :

```
COLUMN_VALUE
```

Exemple : Nested table pour gérer une collection de clés étrangères
Soit la gestion de deux relations pour gérer des livres et des auteurs, avec une relation N:M entre elles. On décide de gérer cette relation par une collection de clés étrangères dans la table des livres.

```
CREATE TABLE tAuteur (
   Id integer PRIMARY KEY,
   Nom char(20),
   Prenom char(20));


CREATE TYPE ListeFkAuteur AS TABLE OF integer;


CREATE TABLE tLivre (
   Titre char(20) PRIMARY KEY,
   fkAuteurs ListeFkAuteur
   ) NESTED TABLE fkAuteurs STORE AS tLivresAuteurs;
```

```
INSERT INTO tAuteur VALUES (1, 'Merle', 'Robert');
INSERT INTO tAuteur VALUES (2, 'Barjavel', 'René');
INSERT INTO tLivre VALUES ('Malvil', ListeFkAuteur(1, 2));
```

```
SELECT t1.Titre, t2.COLUMN_VALUE
FROM tLivre t1, TABLE(t1.fkAuteurs) t2;

TITRE COLUMN_VALUE
----------------------------
Malvil 1
Malvil 2

SELECT t1.Titre, t3.Nom
FROM tLivre t1, TABLE(t1.fkAuteurs) t2, tAuteur t3
WHERE t2.COLUMN_VALUE=t3.Id;

TITRE NOM
----------------------------
Malvil Merle
Malvil Barjavel
```

9. Manipulation d'OID

Une table d'objets, i.e. créée depuis un type, dispose d'un OID pour chacun de ses tuples.

Cet OID est accessible depuis la syntaxe REF dans une requête SQL.

Syntaxe : REF

```
SELECT REF(alias)
FROM nom_table alias
```

Exemple : OID

```
0000280209DB703686EF7044A49F8FA67530383B36853DE7106BC74B6781275ABE5A553A5F01C000340000
```

Syntaxe : Clé étrangère par l'OID

```
CREATE TYPE type_objet AS OBJECT ...

CREATE TABLE table1 OF type_objet ...

CREATE TABLE table2 (
   ...
   clé_étrangère REF type_objet,
   SCOPE FOR (clé_étrangère) IS table1,
   );
```

Remarque : SCOPE FOR

Renforce l'intégrité référentielle en limitant la portée de la référence à une table particulière (alors que REF seul permet de pointer n'importe quel objet de ce type)

Syntaxe : Insertion de données en SQL

```
INSERT INTO table1 (champs1, ..., clé_étrangère)
   SELECT 'valeur1', ..., REF(alias)
   FROM table2 alias
   WHERE clé_table2='valeur';
```

Syntaxe : Insertion de données en PL/SQL

```
DECLARE
   variable REF type_objet;
BEGIN
   SELECT REF(alias) INTO variable
   FROM table2 alias
   WHERE clé_table2='valeur';


   INSERT INTO tCours (champs1, ..., clé_étrangère)
   VALUES ('valeur1', ..., variable);
END;
```

Syntaxe : Navigation d'objets

La référence par OID permet la navigation des objets sans effectuer de jointure, grâce à la syntaxe suivante :

```
SELECT t.clé_étrangère.attribut_table2
FROM table1 t;
```

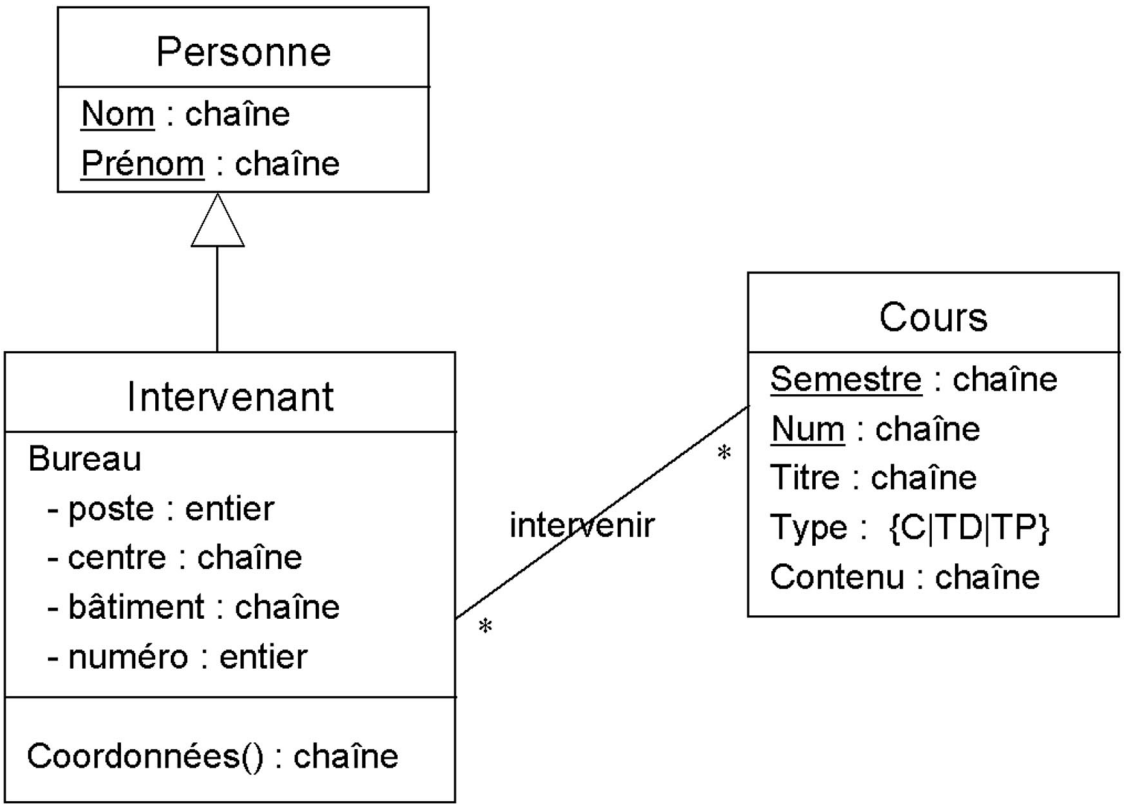
Section B2. Exemples

Objectifs pédagogiques

Savoir déduire un modèle logique relationnel-objet depuis un modèle conceptuel E-A ou UML.
Intégrer les apports du modèle relationnel-objet par rapport au relationnel dans ce processus

1. Gestion de cours

Modèle conceptuel



▲ IMG. 31 : MODÈLE CONCEPTUEL EN UML

Modèle logique

```
Type Bureau : <
   poste:entier,
   centre:chaîne,
   bâtiment:chaîne,
   numéro:entier,
   =coordonnées():chaîne
>


Type Personne : < nom:chaîne, prénom:chaîne >


Type Intervenant hérite de Personne : < bureau:Bureau >


tIntervenant de Intervenant (nom, prénom)


Type RefIntervenant : <nom:chaîne, prénom:chaîne>
Type ListeRefIntervenants: collection de <RefIntervenant>
Type Cours(semestre:chaîne, num:entier, titre:chaîne, type:enum, contenu:chaîne,
lIntervenant:ListeRefIntervenant)


tCours de Cours (semestre, num)
```

Remarque : Méthode coordonnées()

On a choisi de mettre la méthode coordonnées au niveau du type Bureau plutôt que du type Personne, pour augmenter les possibilités de réutilisabilité (car d'autres entités que Intervenant pourraient utiliser le type Bureau). C'est une possibilité offerte par le fait que les attributs composés donnent naissance à des types utilisateurs (on décide donc de remonter certaines méthodes qui ne concerne que cet attribut composé au niveau de ce type).

Implémentation

```
CREATE TYPE Bureau AS OBJECT (
   poste number(4),
   centre char(2),
   batiment char(1),
   numero number(3),
   MEMBER FUNCTION coordonnees RETURN varchar2
   );
CREATE TYPE BODY Bureau
IS
   MEMBER FUNCTION coordonnees RETURN varchar2
   IS
   BEGIN
      RETURN centre||batiment||numero||' - poste '||poste;
   END;
END;


CREATE TYPE Personne AS OBJECT(
   pkNom varchar2(50),
   pkPrenom varchar2(50)
) NOT FINAL ;


CREATE TYPE Intervenant UNDER Personne (
   aBureau Bureau
);


CREATE TABLE tIntervenant OF Intervenant (
   PRIMARY KEY(pkNom,pkPrenom)
);


CREATE TYPE RefIntervenant AS OBJECT (
   nom varchar2(50),
   prenom varchar2(50)
);
CREATE TYPE ListeRefIntervenants AS TABLE OF RefIntervenant;


CREATE TYPE Cours AS OBJECT(
   pkSemestre char(5),
   pkNum number(2),
   aTitre varchar2(50),
   aType char(2),
   aContenu varchar2(300),
   lIntervenant ListeRefIntervenants
);


CREATE TABLE tCours OF Cours (
   PRIMARY KEY(pkSemestre, pkNum),
   CHECK (aType='C' or aType='TD' or aType='TP')
)
NESTED TABLE lIntervenant STORE AS ntCoursIntervenants;
```

Initialisation

```
INSERT INTO tIntervenant VALUES ('CROZAT', 'Stéphane',
Bureau('4287','R','A',108));
INSERT INTO tIntervenant VALUES ('JOUGLET', 'Antoine',
Bureau('4423','R','C',100));


INSERT INTO tCours VALUES ('P2003',1,'Conception','C','E-A et UML',
ListeRefIntervenants(RefIntervenant('CROZAT','Stéphane')));
INSERT INTO tCours VALUES ('P2003',5,'Access','TP','Initiation',
ListeRefIntervenants(RefIntervenant('CROZAT','Stéphane'),
RefIntervenant('JOUGLET','Antoine')));
```

Questions

```
SET LINESIZE 60
SET PAGESIZE 10
COLUMN Quand FORMAT A5
COLUMN Quoi FORMAT A10
COLUMN Qui FORMAT A20
COLUMN Ou FORMAT A19
```

```
SELECT t.pkNom||' '||t.pkPrenom AS Qui, t.aBureau.coordonnees() AS Ou
FROM tIntervenant t;


QUI OU
-------------------------------------
CROZAT Stéphane R A108 - poste 4287
JOUGLET Antoine R C100 - poste 4423
```

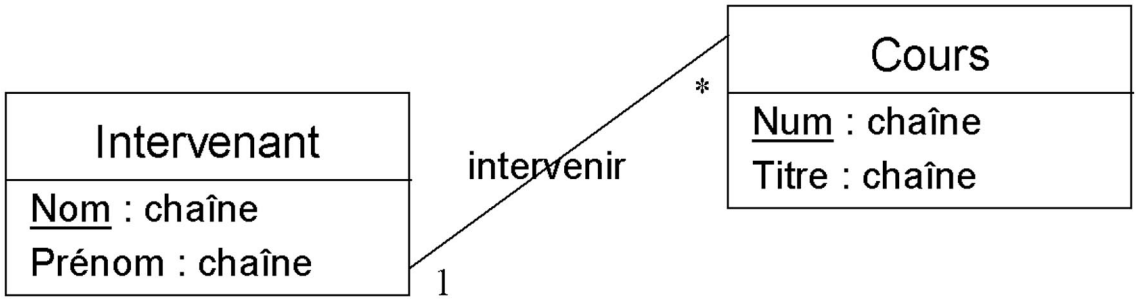
```
SELECT c.pkSemestre AS Quand,c.aTitre AS Quoi,ci.Nom AS Qui
```

```
FROM tCours c, TABLE(c.lIntervenant) ci;


QUAND QUOI QUI
------------------------------
P2003 Conception CROZAT
P2003 Access CROZAT
P2003 Access JOUGLET
```

```
SELECT c.pkSemestre AS Quand,c.aTitre AS Quoi,i.pkNom AS
Qui,i.aBureau.coordonnees() AS Ou
FROM tCours c, TABLE(lIntervenant) ci, tIntervenant i
WHERE ci.nom=i.pkNom AND ci.prenom=i.pkPrenom;


QUAND QUOI QUI OU
-----------------------------------------------
P2003 Conception CROZAT R A108 - poste 4287
P2003 Access CROZAT R A108 - poste 4287
P2003 Access JOUGLET R C100 - poste 4423
```

2. Gestion de cours simplifiée (version avec OID)

Modèle conceptuel



▲ IMG. 32 : MODÈLE CONCEPTUEL EN UML

Modèle logique

```
Type Intervenant : < nom:chaîne, prénom:chaîne >
tIntervenant de Intervenant (nom)


Type Cours(num:entier, titre:chaîne, fkIntervenant REF tIntervenant)
tCours de Cours (num)
```

Implémentation

```
CREATE TYPE Intervenant AS OBJECT (
   pkNom varchar2(50),
   aPrenom varchar2(50)
);


CREATE TABLE tIntervenant OF Intervenant (PRIMARY KEY (pkNom));


CREATE TYPE Cours AS OBJECT (
   pkNum number(2),
   aTitre varchar2(50),
   fkIntervenant REF Intervenant
);


CREATE TABLE tCours OF Cours (
   SCOPE FOR (fkIntervenant) IS tIntervenant,
   PRIMARY KEY(pkNum)
);
```

Initialisation de la table d'objets tIntervenant

```
INSERT INTO tIntervenant
VALUES ('CROZAT', 'Stéphane');
```

Insertion de données dans la table d'objets tCours (SQL)

```
INSERT INTO tCours
   SELECT 2, 'Modélisation', REF(i)
   FROM tIntervenant i
   WHERE pkNom='CROZAT';
```

Insertion de données dans la table tCours (PL/SQL)

```
DECLARE
   ri REF Intervenant;
BEGIN
   SELECT REF(i) INTO ri
   FROM tIntervenant i
   WHERE i.pkNom='CROZAT';


   INSERT INTO tCours
   VALUES (1,'Conception',x);
END;
```

Sélection : Clé étrangère sous forme d'OID

```
SELECT * FROM tCours;


PKNUM ATITRE FKINTERVENANT
------------------------------------------------------------
1 Conception
0000220208DE4BD8F3191044589847E31E64B83EBB5807280A335B44C4AD958EDF20D185B5
2 Modélisation
0000220208DE4BD8F3191044589847E31E64B83EBB5807280A335B44C4AD958EDF20D185B5
```

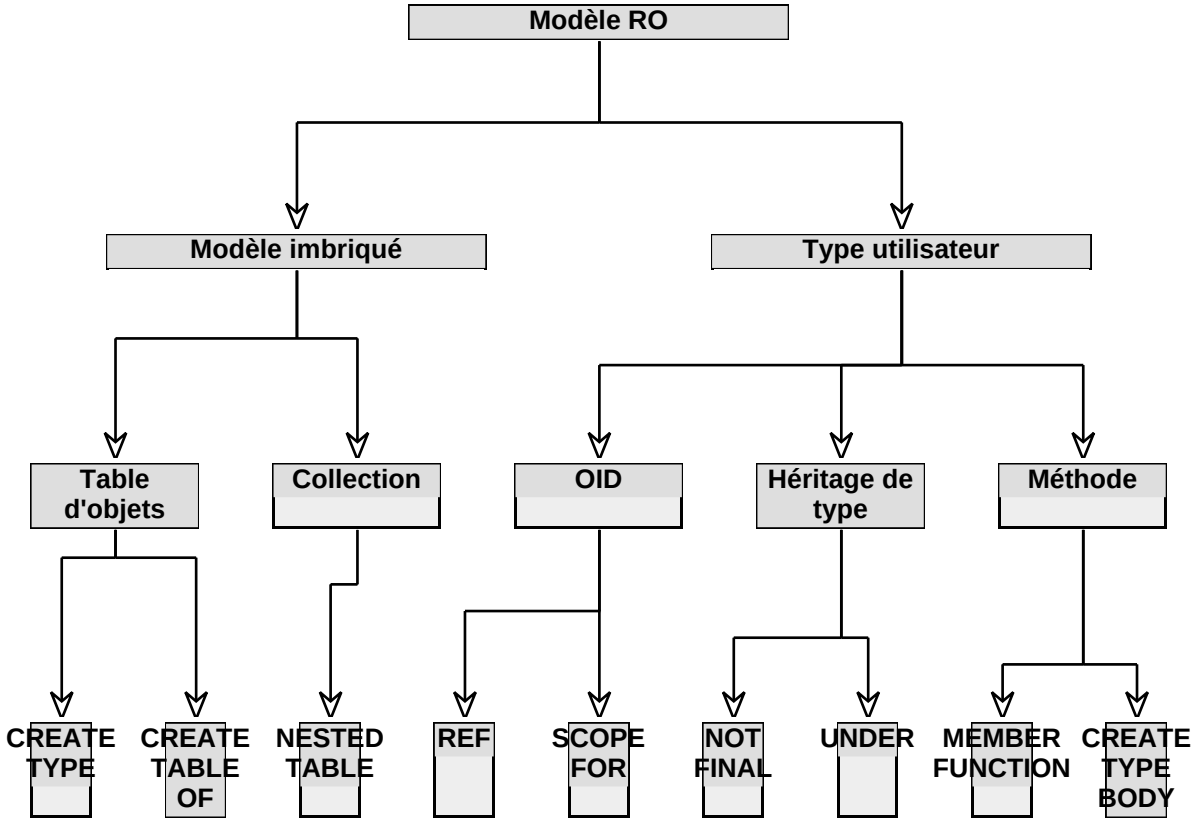
Sélection : Jointure sur l'OID

```
SELECT c.pkNum, i.pkNom
FROM tCours c, tIntervenant i
WHERE c.fkIntervenant=REF(i);


PKNUM PKNOM
-------------------------
1 CROZAT
2 CROZAT
```

Sélection : Navigation d'objets

```
SELECT pkNum, c.fkIntervenant.pkNom
FROM tCours c


PKNUM PKNOM
-------------------------
1 CROZAT
2 CROZAT
```

En résumé...



Pour continuer...

DELMAL P, "SQL2 SQL3, applications à Oracle", De Boeck Université, 2001.

On consultera les chapitre 11 et 12 pour avoir une description des extensions SQL3 et de ses implémentations et extensions sous Oracle (version 8i).

Chapitre C. Questions/réponses sur le relationel-objet

Question/Réponse 1. Association 1:N

Quand on a une association 1:N entre deux classes (par exemple une équipe possède plusieurs athlètes) est-ce qu'il faut : créer un nouvel attribut dans athlète qui soit une référence à l'équipe ou bien créer une nested table d'athlètes dans équipe?

En général il est préférable de transformer comme pour le relationnel, avec une clé étrangère. Il est néanmoins possible d'utiliser des nested tables, mais dans ce cas les tuples de la table côté N deviennent de simples attributs (composites) de la table côté 1 (il ne pourront plus être référencés à leur tour par exemple). Si cela ne pose pas de problème et que cela simplifie l'implémentation, le modèle imbriqué peut être choisi à la place du référencement par la clé étrangère.

Question/Réponse 2. Association M:N

Lorsqu"il y a une association N:M entre deux classes, et que l'on transforme passe en relationnel, il y a une liste de clé étrangères (collection de références) dans une des relations et pas dans l'autre, pourquoi ?

Lorsque l'on choisit une transformation de l'association N:M par une collection de références, la référence ne se fait que dans un sens, au choix, le sens inverse peut-être retrouvé par parcours de la table référençante. Bien sûr, dans ce cas, une relation est favorisée par rapport à l'autre en terme de performance de recherche notamment.
Recopier la référence dans les deux sens n'est pas souhaitable, car elle introduit de la redondance, mais l'on peut néanmoins le faire si les performances l'exige et si l'on contrôle cette redondance.

Question/Réponse 3. Association M:N

Est ce que l'on peut transformer une association M:N comme pour le relationnel, mais en remplaçant les clé étrangères par des références à des OID ?

Oui, c'est une des quatres solutions possibles.

Cours général

VII

Optimisation

Objectifs pédagogiques

Assimiler la problématique de la performance en bases de données
Connaître les grandes classes de solutions technologiques existantes aux problèmes de performance
Connaître et savoir mobiliser les techniques de conception permettant d'optimiser les performances d'une BD

La conception des SGBDR exige qu'une attention particulière soit portée à la modélisation conceptuelle, afin de parvenir à définir des modèles logiques relationnels cohérents et manipulables. De tels modèles relationnels, grâce au langage standard SQL, présentent la particularité d'être implémentables sur toute plate-forme technologique indépendamment de considérations physiques.
Néanmoins l'on sait que dans la réalité, il est toujours nécessaire de prendre en considération les caractéristiques propres de chaque SGBDR, en particulier afin d'*optimiser* l'implémentation. Les optimisations concernent en particulier la question des *performances*, question centrale dans les applications de bases de données, qui, puisqu'elles manipulent des volumes de données importants, risquent de conduire à des temps de traitement de ces données trop longs par rapport aux besoins d'usage.
Chaque SGBDR propose généralement des mécaniques propres pour optimiser les implémentations, et il est alors nécessaire d'acquérir les compétences particulières propres à ces systèmes pour en maîtriser les arcanes. Il existe néanmoins des principes généraux, que l'on retrouvera dans tous les systèmes, comme par exemple les index, les groupements ou les vues matérialisées. Nous nous proposerons d'aborder rapidement ces solutions pour en examiner les principes dans le cadre de ce cours.
Nous aborderons également quelques techniques de conception, qui consistent à revenir sur la structure proposée par l'étape de modélisation logique, pour établir des modèles de données plus aptes à répondre correctement à des questions de performance. La dénormalisation ou le partitionnement en sont des exemples.

Chapitre A. Optimisation du schéma interne

Objectifs pédagogiques

Assimiler la problématique de la performance en bases de données
Connaître les grandes classes de solutions technologiques existantes aux problèmes de performance
Connaître et savoir mobiliser les techniques de conception permettant d'optimiser les performances d'une BD

Section A1. Introduction à l'optimisation des BD

Schéma interne et performances des applications

La passage au schéma interne (ou physique), i.e. l'implémentation du schéma logique dans un SGBD particulier, dépend de considérations pratiques liées aux performances des applications.

Les possibilités d'optimisation des schémas internes des BD dépendent essentiellement des fonctions offertes par chaque SGBD.
On peut néanmoins extraire certains principes d'optimisation des schémas internes suffisamment généraux pour être applicables dans la plupart des cas.

1. Evaluation des besoins de performance

Eléments à surveiller

Pour optimiser un schéma interne, il est d'abord nécessaire de repérer les aspects du schéma logique qui sont succeptible de générer des problèmes de performance.

On pourra citer à titre d'exemple les paramètres à surveiller suivants :
- Taille des tuples
- Nombre de tuples
- Fréquence d'exécution de requêtes
- Complexité des requêtes exécutées (nombre de jointures, etc.)
- Fréquence des mises à jour (variabilité des données)
- Accès concurents
- Distribution dans la journée des accès
- Qualité de service particulière recherchée
- Paramétrabilité des exécutions
- etc.

Evaluation des coûts

Une fois les éléments de la BD à évaluer repérés, il faut mesurer si oui ou non ils risquent de poser un problème de performance.

L'évaluation des performances peut se faire :
- **Théoriquement**

  En calculant le coût d'une opération (en temps, ressources mémoires, etc.) en fonction de paramètres (volume de données, disparité des données, etc.). En général en BD le nombre de paramètres est très grand, et les calculs de coûts trop complexes pour répondre de façon précise aux questions posées.

- **Empiriquement**

  En réalisant des implémentations de parties de schéma et en les soumettant à des tests de charge, en faisant varier des paramètres. Ce modèle d'évaluation est plus réaliste que le calcul théorique. Il faut néanmoins faire attention à ce que les simplifications d'implémentation faites pour les tests soient sans influence sur ceux-ci.

Proposition de solutions

Une fois certains problèmes de performance identifiés, des solutions d'optimisation sont proposées, puis évaluées pour vérifier leur impact et leur réponse au problème posé.

Parmi les solutions d'optimisation existantes, on pourra citer :
- L'indexation

- La dénormalisation
- Le regroupement (*clustering*) de tables
- Le partionnement vertical et horizontal
- Les vues concrètes

2. Indexation

Index

Un index est une structure de données qui permet d'accélérer les recherches dans une table en associant à une clé d'index (la liste des attributs indexés) l'emplacement physique de l'enregistrement sur le disque. Les accès effectuées sur un index peuvent donc se faire sur une liste triée au lieu de se faire par parcours séquentiel des enregistrements.

Cas d'usage

Les index doivent être utilisés sur les tables qui sont fréquemment soumises à des recherches. Ils sont d'autant plus pertinents que les requêtes sélectionnent un petit nombre d'enregistrements (moins de 25% par exemple).
Les index doivent être utilisés sur les attributs :

- souvent mobilisés dans une restriction, donc une jointure [sans que des fonctions leurs soient appliquées]
- très discriminés (c'est à dire pour lesquels peu d'enregistrements ont les mêmes valeurs)
- rarement modifiés

Les index diminuent les performances en mise à jour (puisqu'il faut mettre à jour les index en même temps que les données). Les index ajoutent du volume à la base de données et leur volume peut devenir non négligeable.

Syntaxe Oracle

```
CREATE INDEX nom_index ON nom_table (NomColonneClé1, NomColonneClé2, ...);
```

Remarque : Index implicites

La plupart des SGBD créent un index pour chaque clé (primaire ou candidate). En effet la vérification de la contrainte d'unicité à chaque mise à jour des données justifie à elle seule la présence de l'index.

Attributs indexés utilisés dans des fonctions

Lorsque les attributs sont utilisés dans des restrictions ou des tris par l'intermédiaire de fonctions, l'index n'est généralement pas pris en compte. L'opération ne sera alors pas optimisée. Ainsi par exemple dans le code suivant, le restriction sur X ne sera pas optimisée même s'il existe un index sur X.

```
SELECT *
FROM T
WHERE ABS(X) > 100
```

Cette non prise en compte de l'index est logique dans la mesure où, on le voit sur l'exemple précédent, une fois l'attribut soumis à une fonction, le classement dans l'index n'a plus forcément de sens (l'ordre des X, n'est pas l'ordre des valeurs absolues de X).
Lorsqu'un soucis d'optimisation se pose, on cherchera alors à sortir les attributs indexés des fonctions.
Notons que certains SGBD, tel qu'Oracle à partir de la version 8i, offrent des instructions d'indexation sur les fonctions qui permettent une gestion de l'optimisation dans ce cas particulier (Delmal, 2001).

3. Dénormalisation

La normalisation est le processus qui permet d'optimiser un modèle logique afin de le rendre non redondant. Ce processus conduit à la fragmentation des données dans plusieurs tables.

Dénormalisation

Processus consistant à regrouper plusieurs tables liées par des références, en une seule table, en réalisant statiquement les opérations de jointure adéquates.
L'objectif de la dénormalisation est d'améliorer les performances de la BD en recherche sur les tables considérées, en implémentant les jointures plutôt qu'en les calculant.

Remarque : Dénormalisation et redondance

La dénormalisation est par définition facteur de redondance. A ce titre elle doit être utilisée à bon escient et des moyens doivent être mis en oeuvre pour contrôler la redondance créée.

Quand utiliser la dénormalisation ?

Un schéma doit être dénormalisé lorsque les performances de certaines recherches sont insuffisantes et que cette insuffisance à pour cause des jointures.
La dénormalisation peut également avoir un effet néfaste sur les performances :

- **En mise à jour**

  Les données redondantes devant être dupliquées plusieurs fois.

- **En contrôle supplémentaire**

  Les moyens de contrôle ajoutés (triggers, niveaux applicatifs, etc.) peuvent être très couteux.

- **En recherche ciblée**

  Certaines recherches portant avant normalisation sur une "petite" table et portant après sur une "grande" table peuvent être moins performantes après qu'avant.

4. Groupement de tables

Groupement de table

Un groupement ou *cluster* est une structure *physique* utilisée pour stocker des tables sur lesquelles doivent être effectuées de nombreuses requêtes comprenant des opérations de jointure.
Dans un groupement les enregistrements de plusieurs tables ayant une même valeur de champs servant à une jointure (clé du groupement) sont stockées dans un même bloc physique de la mémoire permanente.
Cette technique optimise donc les opérations de jointure, en permettant de remonter les tuples joints par un seul accès disque.

Clé du groupement

Ensemble d'attributs précisant la jointure que le groupement doit optimiser.

Syntaxe sous Oracle

Il faut d'abord définir un groupement, ainsi que les colonnes (avec leur domaine), qui constituent sa clé. On peut ainsi ensuite, lors de la création d'une table préciser que certaines de ses colonnes appartiennent au groupement.

```
CREATE CLUSTER nom_cluster (NomColonneClé1 type, NomColonneClé2 type, ...);

CREATE TABLE nom_table (...)
   CLUSTER nom_cluster (Colonne1, Colonne2, ...);
```

Remarque

Le *clutering* diminue les performances des requêtes portant sur chaque table prise de façon isolée, puisque les enregistrements de chaque table sont stockés de façon éclatée.

Remarque : Groupement et dénormalisation

Le groupement est une alternative à la dénormalisation, qui permet d'optimiser les jointures sans créer de redondance.

5. Partitionnement de table

Partitionnement de table
Le partitionnement d'une table consiste à découper cette table afin qu'elle soit moins volumineuse, permettant ainsi d'optimiser certains traitement sur cette table.
On distingue :

- Le partitionnement vertical, qui permet de découper une table en plusieurs tables, chacune ne possédant qu'une partie des attributs de la table initiale.
- Le partitionnement horizontal, qui permet de découper une table en plusieurs tables, chacune ne possédant qu'une partie des enregistrements de la table initiale.

Implémentation de projection

Partitionnement vertical
Technique consistant à implémenter deux projections ou plus d'une table T sur des table T1, T2, etc. en répétant la clé de T dans chaque Ti pour pouvoir recomposer la table initiale par *jointure* sur la clé (Gardarin, 1999).

Remarque
Ce découpage équivaut à considérer l'entité à diviser comme un ensemble d'entités reliées par des associations 1:1.

Cas d'usage
Un tel découpage permet d'isoler des attributs peu utilisés d'autres très utilisés, et ainsi améliore les performances lorsque l'on travaille avec les attributs très utilisés (la table étant plus petite).
Cette technique diminue les performances des opérations portant sur des attributs ayant été répartis sur des tables différentes (une opération de jointure étant à présent requise).
Le partitionnement vertical est bien entendu sans intérêt sur les tables comportant peu d'attributs.

Implémentation de restriction

Partitionnement horizontal
Technique consistant à diviser une table T en plusieurs sous-table T1, T2, etc. selon des critères de restriction (Gardarin, 1999) et tel que tous les tuples soit conservés de T. La table T est alors recomposable par *union* sur les Ti.

Cas d'usage
Un tel découpage permet d'isoler des enregistrements peu utilisés d'autres très utilisés, et ainsi améliore les performances lorsque l'on travaille avec les enregistrements très utilisés (la table étant plus petite). C'est le cas typique de l'archivage. Un autre critère d'usage est le fait que les enregistrements soient toujours utilisés selon un partitionnement donné (par exemple le mois de l'année).
Cette technique diminue les performances des opérations portant sur des enregistrements ayant été répartis sur des tables différentes (une opération d'union étant à présent requise).
Le partitionnement horizontal est bien entendu sans intérêt sur les tables comportant peu d'enregistrements.

6. Vues concrètes

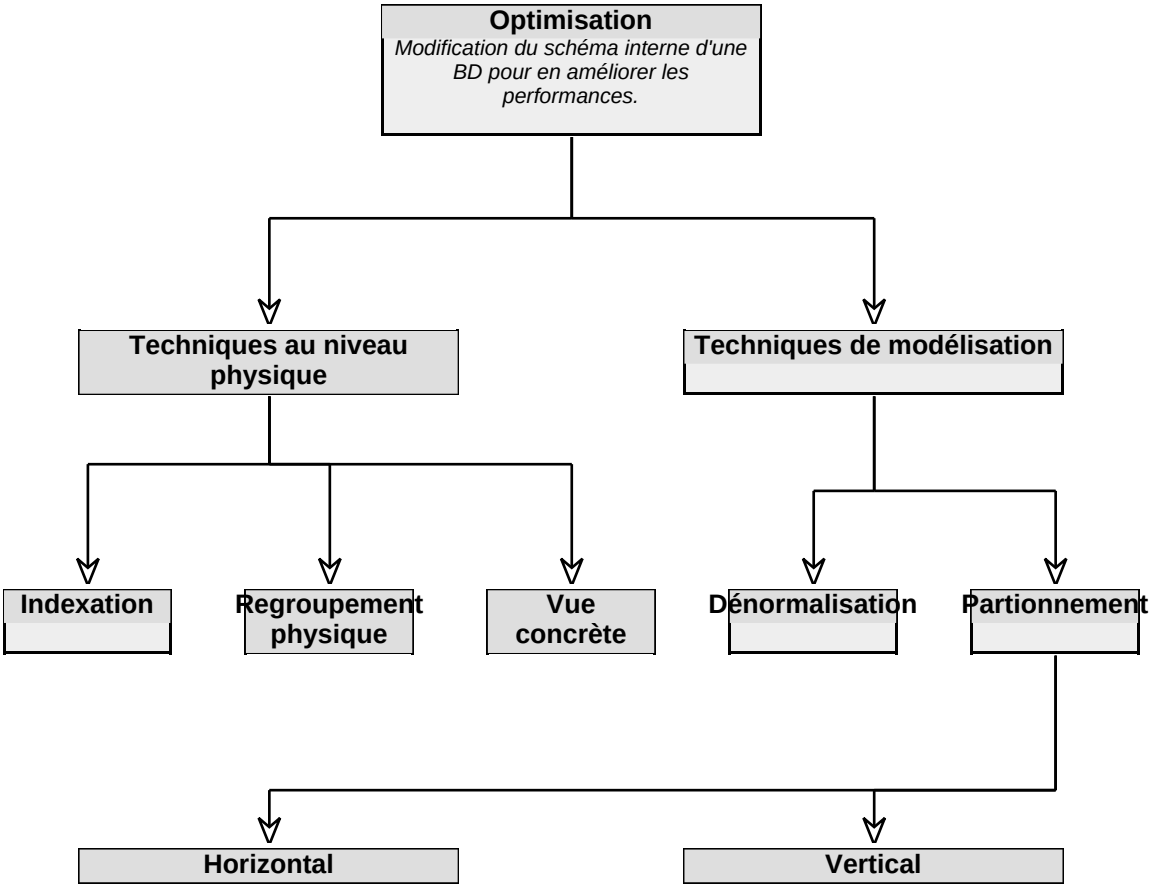
Un moyen de traiter le problème de requêtes dont les temps de calcul sont très longs et les fréquences de mise à jour faible est l'utilisation de vues concrètes.

Vue concrète
Une vue concrète est un stockage statique (dans une table) d'un résultat de requête.

Un accès à une vue concrète permet donc d'éviter de recalculer la requête et est donc aussi rapide qu'un accès à une table isolée. Il suppose par contre que les données n'ont pas été modifiées (ou que leur modification est sans conséquence) entre le moment où la vue a été calculée et le moment où elle est consultée.
Une vue concrète est généralement recalculée régulièrement soit en fonction d'un évènement particulier (une mise à jour par exemple), soit en fonction d'un moment de la journée ou elle n'est pas consultée et où les ressources de calcul sont disponibles (typiquement la nuit).

- *Synonyme : Vue matérialisée.*

En résumé...



Pour aller plus loin...

"http://wwwlsi.supelec.fr/www/yb/poly_bd/tdm.html", BD et SGBD : Table des Matières, **BOURDA Y,** janvier, 2004.

Des informations générales sur les index et sur les clusters.

DELMAL P, "SQL2 SQL3, applications à Oracle", De Boeck Université, 2001.

Une bonne description des index (page 79) et clusters (page 81) par l'image.

Une description intéressante d'une fonction (sous Oracle 8i) qui permet d'indexer des *fonctions*, ce qui répond à un problème important d'optimisation (page 84).
Une description des vues matérialisées, leur implémentation sous Oracle, des exemples d'usage dans le domaine du *data warehouse*.

"http://penarvir.univ-brest.fr/%7Eead/CDL/ORACLE/oracle-node3_mn.html", Accélération des performances et organisation physique, **RIBAUD V,** janvier, 2004.

Un aperçu général de la gestion des index et des clusters sous Oracle 8.

Bibliographie

Ouvrage de référence conseillé

DELMAL P, "SQL2 SQL3, applications à Oracle", De Boeck Université, 2001.

Cet ouvrage aborde tous les thèmes du cours de façon claire et illustrée, en dehors de la phase de modélisation.

Bibliographie

CELKO J, "SQL avancé : Programmation et techniques avancées.", Vuibert, 2000.

CHEN P, "The entity-Relationsheep Model - Towards a Unified View of Data" *in* "ACM Transactions on Database systems", mars 1976, n°.1.

CODD E, "A relational model for large shared data banks" *in* "Communications de l'ACM", juin 1970, n°.6, pp.377-387.

DELMAL P, "SQL2 SQL3, applications à Oracle", De Boeck Université, 2001.

GARDARIN G, "Bases de données : objet et relationnel", Eyrolles, 1999.

MATA-TOLEDO R, CUSHMAN P, "Programmation SQL", Ediscience, 2003.

MULLER P, "Modélisation objet avec UML", Eyrolles, Paris, 1998.

ROQUES P, VALLÉE F, "UML 2 en action : De l'analyse des besoins à la conception J2EE", 385 pages, architecte logiciel, Eyrolles, Paris, 2004.

SOUTOU C, "De UML à SQL : Conception de bases de données", Eyrolles, 2002.

TARDIEU H, ROCHFELD A, COLLETI R, "Méthode MERISE Tome 1 : Principes et outils", Les Editions d'Organisation, Paris, 1983.

TARDIEU H, ROCHFELD A, COLLETI R, PANET G, VAHEE G, "Méthode MERISE Tome 2 : Démarche et pratiques", Les Editions d'Organisation, Paris, 1985.

"http://developpeur.journaldunet.com/dossiers/alg_uml.shtml", UML en 5 étapes, **MORLON J,** avril, 2004.

"http://developpeur.journaldunet.com/tutoriel/cpt/031013cpt_uml5conseils.shtml", Cinq petits conseils pour un schéma UML efficace, **BORDERIE X,** avril, 2004.

"http://eric.univ-lyon2.fr/~jdarmont/docs/sise-bd-ex_rel.pdf", Exercices - Modèle Relationnel, **DARMONT J,** janvier, 2004.

"http://eric.univ-lyon2.fr/~jdarmont/docs/sise-bd-ex_uml.pdf", Exercices - Modèle UML, **DARMONT J,** janvier, 2004.

"http://eric.univ-lyon2.fr/~jdarmont/docs/sise-bd-exam0203.pdf", Examen de bases de données, **DARMONT J,** janvier, 2004.

"http://eric.univ-lyon2.fr/~jdarmont/docs/sise-bd-td1isea.pdf", Relationnel-objet, Relationnel étendu, **DARMONT J,** janvier, 2004.

"http://eric.univ-lyon2.fr/~jdarmont/docs/sise-bd-td2.pdf", TD2 : Oracle SQL, **DARMONT J,** janvier, 2004.

"http://eric.univ-lyon2.fr/~jdarmont/tutoriel-sql/", Tutoriel SQL, **DARMONT J,** janvier, 2004.

"http://penarvir.univ-brest.fr/%7Eead/CDL/ORACLE/oracle-node3_mn.html", Accélération des performances et organisation physique, **RIBAUD V,** janvier, 2004.

"http://www.developpez.com/hcesbronlavau/Transactions.htm", Les transactions, **CESBRON LAVAU H,** janvier, 2004.

"http://www.info.uqam.ca/~godin/DiaposParChap/ChapI-5.ppt", Introduction au modèle relationnel, **GODIN R,** avril, 2004.

"http://www.sybase.com/products/enterprisemodeling", Sybase PowerDesigner, septembre, 2002.

"http://www-inf.int-evry.fr/COURS/BD/BD_REL/SUPPORT/poly.html#RTFToC30", Contrôle des accès

concurrents et reprise, **DEFUDE B,** janvier, 2004.

"http://wwwlsi.supelec.fr/www/yb/poly_bd/tdm.html", BD et SGBD : Table des Matières, **BOURDA Y,** janvier, 2004.

"uml.free.fr", UML en Français, septembre, 2002.

"www.objecteering.com", Objecteering software, septembre, 2002.

Glossaire

API

Une API est une définition d'un ensemble de fonctions ou méthodes (en programmation objet) dont l'appel paramétré permet de piloter une application. Une API permet donc d'utiliser un librairie informatique sans avoir besoin d'en connaître le fonctionnement interne.

Constructeur d'objet

En programmation orientée objet, un constructeur d'objet est une méthode particulière d'une classe qui permet d'instancier un objet de cette classe. L'appel à cette méthode de classe a donc pour conséquence la création d'un nouvel objet de cette classe.

Exception

Une exception est un évènement généré par un système informatique pour signifier une erreur d'exécution. La gestion des exceptions est un aspect de la programmation informatique, qui consiste à intercepter ces évènements particuliers et à les traiter pour, soit les corriger automatiquement, soit en donner une information appropriée à un utilisateur humain.

Impedance mismatch

Le terme d'impedance mismatch renvoie au décalage qui peut exister entre le niveau d'abstraction de deux langages qui ont à travailler sur des structures de données communes, par exemple un langage applicatif objet et un langage de données relationnel. L'impedance mismatch a des conséquences négatives en terme de complexification de l'implémentation et en terme de performance, puisqu'il faut constamment passer d'une strucuture de données à l'autre.

RAID

La techologie RAID pemet de répartir de l'information à stocker sur plusieurs "petits" disques, au lieu de la concentrer sur un seul "gros" disque. Cette technologie permet donc d'améliorer les performances (les accès disques pouvant être parallélisés) et d'améliorer la sureté (en répartissant les risques de crash et en jouant sur une redondance des données). Il existe plusieurs types d'architecture RAID, privilégiant ou combinant la parallélisation et la redondance.

Signification des sigles

1NF	Première Forme Normale
2NF	Deuxième Forme Normale
3NF	Troisième Forme Normale
4NF	Quatrième Forme Normale
5NF	Cinquième Forme Normale
ACID	Atomique, Cohérent, Isolé, Durable
API	Application Program Interface
BCNF	Forme Normale de Boyce-Codd
BD	Base de Données
DF	Dépendance Fonctionnelle
DFE	Dépendance Fonctionnelle Elémentaire
E-A	Entité-Association
E-R	Entity-Relationship
FIFO	First In First Out
IHM	Interface Homme Machine
LCD	Langage de Contrôle de Données
LDD	Langage de Définition de Données
LMD	Langage de Manipulation de Données
MCD	Modèle Conceptuel de Données
MLD	Modèle Logique de Données
OID	Object Identifier
OMG	Object Management Group
SGBD	Système de Gestion de Bases de Données
SGBDOO	Système de Gestion de Bases de Données Orientées Objets
SGBDR	Système de Gestion de Bases de Données Relationnelles
SGBDRO	Système de Gestion de Bases de Données Relationnelles-Objets
SGF	Système de Gestion de Fichiers
SI	Système d'Information
SQL	Structured Query Language
UML	Unified Modeling Language

Index

Fiches mémo

Vue d'ensemble

En quoi une base de données est-elle plus intéressante qu'un système de fichier classique ?

Quelles sont les fonctions remplies par un SGBD ?

Notions générales

Pourquoi est-ce que l'on distingue trois niveaux de modélisation lors de la conception d'une base de données ?

Quelles est la différence entres le schéma conceptuel et le ou les schémas externes ?

Quelles sont les fonctions d'un langage orienté données ?

A quoi et à qui sert un dictionnaire de données ?

Les méthodes de conception de bases de données

Pourquoi est-il fondamental mais difficile de parvenir à un MCD correct ?

Enoncer quelques actions à mener pour réaliser une spécification générale de l'existant et des besoins ?

Qu'est ce qui différencie fondamentalement un MCD d'un MLD ?

Le modèle E-A

Enoncer les principaux éléments composants le modèle E-A ?

Quels sont les avantages apportés par l'extension du modèle E-A ?

Que permet d'exprimer une entité de type faible ?

Les diagrammes de classes UML

Quels sont les principaux éléments du diagramme de classes UML ?

Quelles sont les différences et points communs entre la diagramme de classe UML et le modèle E-A étendu ?

Description du modèle relationnel

Qu'est ce qu'un domaine ?

Quel rapport y-a-t il entre une relation et une table ?

Comment identifie-t-on un attribut d'une relation ?

Comment identifie-t-on un enregistrement d'une relation ?

Quelle problème pose la redondance et comment le résoudre ?

Le passage E-A vers Relationnel

Le passage E-A ou UML vers relationnel est-il systématique ou soumis à interprétation ? Pourrait-il être réalisé par un algorithme ?

Est ce que l'un des deux modèles conceptuels, E-A ou UML, est plus adapté au passage au relationnel ?

Pourquoi dispose-t-on de trois méthodes pour traduire l'héritage dans un modèle relationnel ? Ces trois méthodes sont-elles équivalentes ?

Le passage UML vers Relationnel

Le passage UML vers relationnel est-il systématique ou soumis à interprétation ? Pourrait-il être réalisé par un algorithme ?

Pourquoi dispose-t-on de trois méthodes pour traduire l'héritage dans un modèle relationnel ? Ces trois méthodes sont-elles équivalentes ?

Algèbre relationnelle

Quels sont les opérateurs algébriques de base ? Quels sont les autres opérateurs ? Qu'est ce qui les différencie ?

Quels sont les opérateurs ensemblistes ? Qu'est ce qui les caractérise ?

Pourquoi la jointure est-elle un opérateur essentiel ?

Qu'est ce qui différencie une jointure externe d'une jointure classique ?

Le LMD de SQL

A quoi sert le LMD ?

Quel rapport y-a-t il entre le SQL et l'algèbre relationnelle ?

Pourquoi SQL n'est-il pas un langage de programmation ?

Théorie de la normalisation relationnelle

En quoi peut-on dire que certains schémas relationnels sont mauvais ?

Pourquoi est-il primordial de repérer les dépendances fonctionnelles sur un schéma relationnel ?

Comment repère-t-on ces dépendances fonctionnelles ?

Que sont les axiomes d'Armstrong et à quoi servent-ils ?

Qu'est ce que la décomposition d'une relation ?

Pourquoi le respect de la première forme normale reste-t-il en partie subjectif ?

Quelle forme normale est généralement souhaitable pour un schéma relationnel ?

Le LDD de SQL

A quoi sert le LDD ?

En quoi le LDD est il un langage déclaratif ?

Quel rapport y-a-t il entre le LDD et le concept de relation ?

Le LCD de SQL

Quels types de droits peuvent être accordés ou révoqués en SQL ?

Pourquoi peut-on dire que la gestion des droits est décentralisée en SQL ?

Transactions

Pourquoi une transaction est-elle atomique ?

Pourquoi une transaction est-elle cohérente ?

Pourquoi une transaction est-elle isolée ?

Pourquoi une transaction est-elle durable ?

Fiabilité

A quoi sert le journal des transactions ?

Qu'est ce qu'un point de contrôle ?

L'aglorithme de reprise UNDO-REDO terminera-t-il toutes les transactions qui étaient commencées au moment de la panne ?

Concurrence

Laquelle des propriétés ACID des transactions est-elle particulièrement utile pour gérer les accès concurrents ?

Le verrouillage est-il une solution parfaite pour la gestion de la concurrence ?

Illustrations sous Oracle

Pourquoi peut-on dire que les transactions sont les unité logique de travail, les unités d'intégrité, les unités de reprise et les unités de concurrence ?

Introduction : R, OO, RO

Quels sont les atouts du modèle relationnel-objet par rapport au modèle relationnel ?

Le modèle relationnel-objet

Quels sont les extensions les plus importantes apportés par le modèle relationnel-objet au modèle relationnel ?

En quoi le modèle relationnel-objet peut-il être considéré comme plus proche que le modèle relationnel du modèle conceptuel ?

.
.
.
.
.

Mapping conceptuel vers relationnel-objet

En quoi le mapping E-A vers relationnel-objet est-il plus fidèle que le mapping E-A vers relationnel ?

.
.
.
.
.

Dans quel cas l'utilisation de collections peut-il être destructeur de sémantique ?

.
.
.
.
.

En quoi l'implémentation de l'héritage en relationnel-objet simplifie-t-il le mapping ? En quoi ne le simplifie-t-il pas ?

.
.
.
.
.

SQL3 (implémentation Oracle 9i)

Qu'apporte les OID par rapport aux clés étrangères classiquement manipulées dans le modèle relationnel ?

.
.
.
.
.

Quelles solutions de modélisation apporte les instructions de création de type sous Oracle ?

.
.
.
.
.

Exemples

L'utilisation d'OID pour les références est-elle préférable à l'utilisation de clés étrangères classique, en particulier dans le cas où la clé étrangère est composée ?

.
.
.
.
.

Introduction à l'optimisation des BD

Citer des paramètres propres à une BD que l'on doit surveiller dans le cadre de la performance ?

Peut-on anticiper sur des problèmes de performance futurs lors de la conception d'une BD ?

Pourquoi n'indexe-t-on pas tous les champs d'une BD ?

Quels problèmes pose la dénormalisation ?

Quels problèmes pose le clustering ?

Quels problèmes pose le partionnement ?

Quels problèmes pose les vues concrètes ?