

Conception de bases de données I

Version 12.01

Cours (Fondamentaux)



STÉPHANE CROZAT

Table des matières

Introduction	6
1 - Présentation des bases de données	7
1.1 BD et SGBD : vue d'ensemble.....	7
1.1.1 Qu'est ce qu'une BD ?.....	7
1.1.2 Qu'est ce qu'un SGBD ?.....	7
1.1.3 Pourquoi des SGBD ?.....	7
1.1.4 Caractéristiques des SGBD.....	8
1.2 Notions générales pour les bases de données.....	9
1.2.1 Notion de données.....	9
1.2.2 Notion de modèle de données.....	9
1.2.3 Notion de schéma de données.....	10
1.2.4 Notion de langage de données.....	10
1.2.5 Notion d'administration de données.....	11
1.3 Les méthodes de conception de bases de données.....	11
1.3.1 Méthodologie de conception d'une base de données.....	11
1.3.2 La méthode MERISE et le modèle E-A.....	12
1.3.3 Le langage de modélisation UML.....	13
1.3.4 Éléments pour l'analyse de l'existant et des besoins.....	13
1.3.5 Le MCD.....	14
1.3.6 Le MLD.....	14
1.4 En résumé : Conception de bases de données.....	15
2 - Le niveau conceptuel : la modélisation des bases de données	16
2.1 Bases du diagramme de classes UML.....	16
2.1.1 Présentation d'UML.....	16
2.1.2 Classes.....	16
2.1.3 Attributs.....	17
2.1.4 Méthodes.....	18
2.1.5 Associations.....	18
2.1.6 Cardinalité.....	19
2.1.7 Héritage.....	20
2.1.8 Exemple : Des voitures et des conducteurs.....	21
2.2 Diagramme de classes UML avancé.....	22
2.2.1 Explicitation des associations.....	22
2.2.2 Classe d'association.....	22
2.2.3 Associations ternaires.....	23
2.2.4 Composition.....	24
2.2.5 Aggrégation.....	25
2.2.6 Classes abstraites.....	26
2.2.7 Contraintes.....	26
2.2.8 Contraintes sur les associations.....	28
2.2.9 Paquetages.....	28
2.2.10 Stéréotype.....	29
2.3 Le modèle E-A.....	30
2.3.1 Le modèle E-A en bref.....	30
2.3.2 Entité.....	30
2.3.3 Association.....	31
2.3.4 Cardinalité d'une association.....	31
2.3.5 Modèle E-A étendu.....	32
2.3.6 Entité de type faible.....	33
2.3.7 Illustration d'entités faibles.....	33
2.4 En résumé : Schéma conceptuel.....	34
2.5 Bibliographie commentée sur la modélisation UML.....	34
3 - Le niveau logique : la modélisation relationnelle	36
3.1 Description du modèle relationnel.....	36
3.1.1 Le niveau logique.....	36
3.1.2 Le modèle relationnel.....	36
3.1.3 Domaine.....	36
3.1.4 Produit cartésien.....	37
3.1.5 Relation.....	37
3.1.6 Attribut et enregistrement.....	37
3.1.7 Exemple : La relation "Vol".....	38
3.1.8 Clé.....	38
3.1.9 Clé artificielle.....	39
3.1.10 Lien entre relations.....	39

3.1.11 Éclatement des relations pour éviter la redondance.....	40
3.1.12 Clé étrangère.....	41
3.1.13 Schéma relationnel.....	42
3.1.14 Exemple de schéma relationnel pour la géographie.....	42
3.2 Passage UML-Relationnel : Cas systématiques.....	43
3.2.1 Transformation des classes.....	43
3.2.2 Transformation des attributs.....	44
3.2.3 Transformation des attributs dérivés et méthodes.....	45
3.2.4 Transformation des associations 1:N.....	45
3.2.5 Transformation des associations N:M.....	46
3.2.6 Transformation des classes d'association.....	47
3.2.7 Transformation des agrégations.....	49
3.2.8 Transformation des compositions.....	49
3.2.9 Liste des contraintes.....	51
3.2.10 Correspondance entre UML et relationnel.....	51
3.3 Passage UML-Relationnel : Associations 1:1.....	51
3.3.1 Transformation des associations 1:1 (cas général).....	52
3.3.2 Transformation des associations 1..1:1..1.....	52
3.3.3 Transformation des associations 0..1:1..1.....	53
3.3.4 Transformation des associations 0..1:0..1.....	53
3.3.5 Exemple de choix pour une relation 1:1.....	54
3.4 Passage UML-Relationnel : Héritage.....	54
3.4.1 Transformation de la relation d'héritage.....	54
3.4.2 Transformation de la relation d'héritage par référence.....	56
3.4.3 Transformation de la relation d'héritage par les classes filles.....	57
3.4.4 Transformation de la relation d'héritage par la classe mère.....	58
3.4.5 Éléments de choix.....	59
3.4.6 Les cas simples.....	60
3.4.7 Exemple de transformation d'une relation d'héritage.....	63
3.4.8 Transformations de l'héritage : Synthèse.....	64
3.5 Le passage E-A vers Relationnel.....	65
3.5.1 Transformation des entités.....	65
3.5.2 Transformation des associations.....	66
3.5.3 Transformation des attributs.....	66
3.5.4 Exemple de passage E-A vers relationnel.....	67
3.5.5 Transformation de la relation d'héritage.....	67
3.5.6 Exemple de transformation d'une relation d'héritage.....	67
3.5.7 Comparaison des modèles E-A, UML et relationnel.....	68
3.6 Algèbre relationnelle.....	69
3.6.1 Concepts manipulatoires.....	69
3.6.2 Opérateurs ensemblistes.....	69
3.6.3 Projection.....	70
3.6.4 Restriction.....	70
3.6.5 Produit.....	71
3.6.6 Jointure.....	71
3.6.7 Jointure naturelle.....	72
3.6.8 Jointure externe.....	72
3.6.9 Division.....	73
3.6.10 Proposition de notations.....	73
3.6.11 Exercice de synthèse : Opérateurs de base et additionnels.....	74
3.7 En résumé : Schéma relationnel.....	74
3.8 Bibliographie commentée sur le modèle relationnel.....	74
4 - Le niveau physique : Le langage SQL	75
4.1 Qu'appelle-t-on SQL?.....	75
4.2 Le Langage de Définition de Données de SQL.....	76
4.2.1 Types de données.....	76
4.2.2 Création de tables.....	76
4.2.3 Contraintes d'intégrité.....	77
4.2.4 Exemple de contraintes d'intégrité.....	78
4.2.5 Création de vues.....	79
4.2.6 Suppression d'objets.....	79
4.2.7 Modification de tables.....	79
4.2.8 Exemple de modifications de tables.....	80
4.3 Gestion avec le Langage de Manipulation de Données de SQL.....	81
4.3.1 Insertion de données.....	81
4.3.2 Mise à jour de données.....	81
4.3.3 Suppression de données.....	82
4.4 Questions avec le Langage de Manipulation de Données de SQL.....	82
4.4.1 Sélection.....	82
4.4.2 Opérateurs de comparaisons et opérateurs logiques.....	83
4.4.3 Expression du produit cartésien.....	84
4.4.4 Expression d'une projection.....	84
4.4.5 Expression d'une restriction.....	85
4.4.6 Expression d'une jointure.....	85
4.4.7 Expression d'une jointure externe.....	86
4.4.8 Opérateurs ensemblistes.....	88
4.4.9 Tri.....	89
4.4.10 Fonctions de calcul.....	89

4.4.11 Agrégats.....	90
4.5 Instructions avancées pour le LMD de SQL.....	91
4.5.1 Requêtes imbriquées.....	91
4.5.2 Sous-requête d'existence IN.....	91
4.5.3 Sous-requête d'existence EXISTS.....	92
4.5.4 Sous-requête de comparaison ALL.....	92
4.5.5 Sous-requête de comparaison ANY.....	93
4.5.6 Raffinement de questions dans la clause FROM.....	93
4.6 Le Langage de Contrôle de Données de SQL.....	93
4.6.1 Attribution de droits.....	94
4.6.2 Révocation de droits.....	94
4.6.3 Création d'utilisateurs.....	95
4.7 En résumé : SQL.....	95
4.8 Bibliographie commentée sur le SQL.....	95
5 - La théorie de la normalisation relationnelle	97
5.1 Redondance et normalisation.....	97
5.1.1 Exercice introductif : Redondance.....	97
5.1.2 Les problèmes soulevés par une mauvaise modélisation.....	97
5.1.3 Principes de la normalisation.....	98
5.2 Les dépendances fonctionnelles.....	98
5.2.1 Dépendance fonctionnelle.....	98
5.2.2 Les axiomes d'Armstrong.....	99
5.2.3 Autres propriétés déduites des axiomes d'Armstrong.....	99
5.2.4 DF élémentaire.....	100
5.2.5 Notion de fermeture transitive des DFE.....	100
5.2.6 Notion de couverture minimale des DFE.....	101
5.2.7 Notion de graphe des DFE.....	101
5.2.8 Définition formelle d'une clé.....	101
5.3 Les formes normales.....	102
5.3.1 Formes normales.....	102
5.3.2 Principe de la décomposition.....	102
5.3.3 Première forme normale.....	102
5.3.4 Deuxième forme normale.....	103
5.3.5 Troisième forme normale.....	104
5.3.6 Forme normale de Boyce-Codd.....	105
5.4 Conception de bases de données normalisées.....	105
5.4.1 Processus de conception d'une base de données normalisée.....	105
5.4.2 Algorithme de décomposition.....	106
5.4.3 Exemple de décomposition.....	106
5.5 Bibliographie commentée sur la normalisation.....	107
6 - Technologie Web : Architecture LAPP	108
6.1 Architecture Web.....	108
6.1.1 Notions d'architecture client-serveur.....	108
6.1.2 Notions d'architecture 3-tier.....	109
6.1.3 Notions de serveur Web.....	111
6.1.4 Notion d'architecture Web.....	111
6.1.5 Architecture LAAP.....	112
6.2 Introduction à HTML.....	112
6.2.1 HTML.....	112
6.2.2 XHTML.....	113
6.2.3 Structure générale XHTML.....	113
6.2.4 Balises de base XHTML.....	113
6.2.5 Introduction à CSS.....	114
6.2.6 Mise en ligne.....	115
6.2.7 Formulaires HTML.....	115
6.3 Introduction à PHP.....	115
6.3.1 Présentation de PHP.....	116
6.3.2 Principes de PHP.....	116
6.3.3 Syntaxe PHP.....	117
6.3.4 Variables en PHP.....	117
6.3.5 Structures de contrôle en PHP.....	118
6.3.6 Boucles en PHP.....	118
6.3.7 Fonctions en PHP.....	118
6.3.8 Objets en PHP.....	118
6.3.9 Envoi de texte au navigateur.....	119
6.3.10 Formulaires HTML et PHP.....	120
6.3.11 Variables de session.....	121
6.4 Introduction à PostgreSQL.....	122
6.4.1 Présentation.....	122
6.4.2 Types de données.....	122
6.4.3 Le client textuel "psql".....	122
6.4.4 Exécuter un fichier SQL.....	123

6.4.5 Fichier CSV.....	123
6.4.6 Importer un fichier CSV.....	124
6.4.7 Le client graphique "pgAdminIII"	124
6.4.8 Notion de schéma.....	126
6.5 PHP et BD.....	128
6.5.1 Interfaçage avec PostgreSQL.....	129
6.5.2 Interfaçage PHP avec MySQL.....	129
6.5.3 Interfaçage PHP avec Oracle.....	129
6.5.4 Architecture PHP/Oracle.....	131
6.6 Rappels Linux.....	131
6.6.1 Rappels architecture UTC.....	131
6.6.2 Rappels Unix/Linux.....	131
6.7 Bases de données et applications.....	132
6.7.1 Architecture générale d'une application de base de données.....	132
6.7.2 Méthode générale d'accès à une BD en écriture par un langage de programmation.....	133
6.7.3 Méthode générale d'accès à une BD en lecture par un langage de programmation.....	135
6.8 Bibliographie commentée sur les architectures Web et PHP.....	137
7 - Technologie Access	138
7.1 Introduction à Access.....	138
7.1.1 Présentation d'Access.....	138
7.1.2 Avantages et inconvénient d'Access.....	140
7.1.3 Séparation base de données et application.....	140
7.2 Création de schéma relationnel sous Access.....	141
7.2.1 LDD et création de tables sous Access.....	141
7.2.2 Domaines et types de données sous Access.....	141
7.2.3 Contraintes.....	141
7.2.4 Vues et "requêtes" LDD.....	142
7.3 Le langage de requêtes sous Access.....	143
7.3.1 Questions SQL.....	143
7.3.2 Manipulation des données en QBE.....	143
7.3.3 Clause GROUP BY en QBE.....	143
7.4 Création d'application Access (formulaires et macros).....	144
7.4.1 Formulaire liés à une table.....	144
7.4.2 Formulaires indépendants.....	145
7.4.3 Contrôles listes.....	146
7.4.4 Macros.....	147
7.5 Modules de programmation VBA sous Access.....	149
7.5.1 Structure d'un programme VBA.....	149
7.5.2 Fonctions à connaître.....	149
7.5.3 Objets VBA pour accéder à la BD.....	149
7.5.4 Exemple : Normaliser des chaînes de caractères.....	150
7.5.5 Exemple : Accéder à un fichier externe.....	150
7.5.6 Exemple : Parcourir une table ou une requête stockée.....	150
7.5.7 Exemple : Parcourir une table passée en paramètre.....	150
7.5.8 Exemple : Parcourir une table et gérer les erreurs.....	151
7.5.9 Exemple : Exécuter une requête.....	151
7.5.10 Exemple : Créer un schéma de BD et initialiser les données.....	151
7.5.11 Aide et déboguage.....	151
7.6 Autres aspects.....	152
7.6.1 Gestion des droits.....	152
7.6.2 Run-time.....	152
7.7 En résumé : Access.....	152
7.8 Bibliographie commentée sur Access.....	152
7.9 Questions-réponses sur Access.....	152
Glossaire	154
Signification des abréviations	155
Bibliographie	156
Webographie	157
Index	158

Introduction

Les BD 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 est une couche technologique, idéalement indépendante des implémentations des SGBDR, 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 de façon centralisée 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 renouvelées.

Les BD¹ ont été créées pour faciliter la gestion qualitative et quantitative des données informatiques. Les SGBD¹ sont des applications informatiques permettant de créer et de gérer des BD (comme Oracle ou MySQL par exemple). Les BD **relationnelles** sont les plus répandues et l'on utilise des SGBDR¹ pour les implémenter. Le langage SQL¹ est le langage commun à tous les SGBDR, ce qui permet de concevoir des BD relativement indépendamment des systèmes utilisés.

1.1 BD et SGBD : vue d'ensemble

Objectifs

Comprendre l'intérêt des BD.
Comprendre ce qu'est un SGBD.

1.1.1 Qu'est ce qu'une BD ?



Définition : Base de données

Une BD¹ 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, interrogables 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.

1.1.2 Qu'est ce qu'un SGBD ?



Définition : Système de Gestion de Bases de Données

Un SGBD¹ 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).



Exemple : 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.
- PostgreSQL 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.

1.1.3 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 cœur 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 marketing et commerciales
- Les données relatives au personnel
- etc.

1.1.4 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

Fondamental

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-informatiens**
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 ou des langages assertionnels simples.
- **Administration facilitée des données**
Le SGBD fournit un ensemble d'outils (dictionnaire de données, audit, tuning, statistiques, etc.) pour améliorer les performances et optimiser les stockages.
- **Optimisation de l'accès aux données**
Les temps de réponse et de débits globaux sont optimisés en fonction 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 leur sont imposées.
- **Partageabilité des données**
Les données sont simultanément 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 persistance des données, même en cas de panne, est assurée, grâce typiquement à des sauvegardes et des journaux qui gardent une trace persistante des opérations effectuées.

1.2 Notions générales pour les bases de données

Objectifs

Connaître les différences entre niveau conceptuel, niveau logique et niveau physique.
Comprendre l'importance de la modélisation conceptuelle.

1.2.1 Notion de données



Définition : Données

Élément effectif, réel, correspondant à une type de données.

Synonymes : Occurrence, Instance



Exemple : Données

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



Définition : Type de données

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

Synonymes : Classe



Exemple : Type de données

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

1.2.2 Notion de modèle de données



Définition : Modèle de données

Ensemble de concepts et de règles de composition de ces concepts permettant de décrire des données (cf. Bases de données : objet et relationnel [Gardarin99]).

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 niveau conceptuel**
Il permet de décrire le réel selon une approche ontologique, sans prendre en compte les contraintes techniques.
- **Le niveau 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 niveau physique**
Il correspond aux choix techniques, en terme de SGBD choisi et de sa mise en œuvre (programmation, optimisation, etc.).



Exemple : Exemple de formalisme de modélisation conceptuelle

- Le modèle Entité-Association (cf. The Entity-Relationship Model [Chen76]) a été le plus répandu 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 : Exemple de formalisme de modélisation logique

- Le modèle CODASYL, antérieur au modèle relationnel est un modèle hiérarchique (Tardieu83 [Tardieu83]:193).
- Le modèle relationnel (tabulaire) est le modèle dominant aujourd'hui.
- Le modèle relationnel-objet (adaptation des modèles relationnel et objet au cadre des SGBD) est actuellement en croissance.
- Le modèle objet "pur" reste majoritairement au stade expérimental et de la recherche.
- D'autres modèles (arborescent, réseau, ...) sont surtout utilisés dans des contextes très spécifiques.

1.2.3 Notion de schéma de données



Définition : 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'occurrence 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

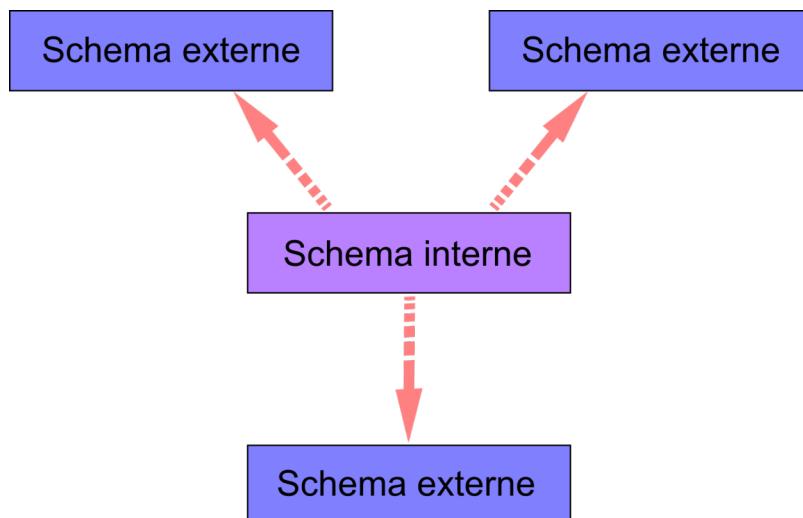
```
Etudiant (172, 'Dupont', 'Lille')
Etudiant (173, 'Durand', 'Paris')
Etudiant (174, 'Martin', 'Orléans')
Module(1, 'SGBD')
Module(1, 'Systèmes d'exploitation')
Inscription(172, 1, 2002)
Inscription(172, 2, 2002)
Inscription(173, 1, 2001)
Inscription(174, 2, 2002)
```



Complément: Vue canonique et vue externe

Le schéma de données peut se décliner selon deux niveaux de représentation :

1. Le niveau interne : Il s'agit du modèle global, formé de l'union des sous-modèles du niveau externe.
2. Le niveau externe : Il s'agit de sous-ensembles du modèle global, vus par des groupes d'utilisateurs particuliers (on parle d'ailleurs de "vue" pour un modèle externe).



Schémas interne et externes

1.2.4 Notion de langage de données



Définition : Langage de données

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

Synonymes : 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 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 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 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.

1.2.5 Notion d'administration de données

 Définition : 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.

 Définition : 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

1.3 Les méthodes de conception de bases de données

Objectifs

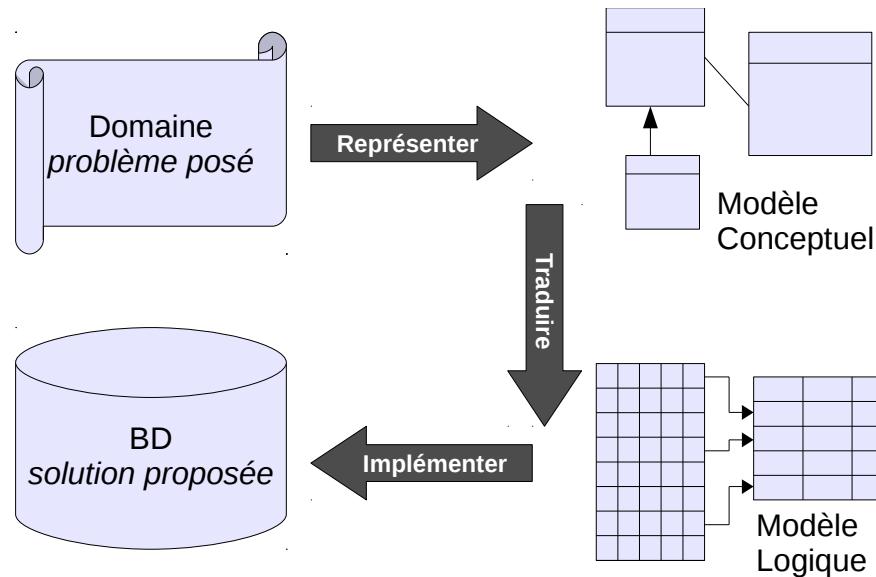
Connaître la méthodologie de conception d'une BD.

1.3.1 Méthodologie de conception d'une base de données

 Méthode : Étapes 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



Graphique 1 Processus de conception d'une base de données

Fondamental

- **Bien analyser** le problème posé en amont
- **Bien modéliser** le problème au niveau conceptuel avant de passer aux niveaux logiques et physiques

Conseil : 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 modélisation conceptuelle 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.

Conseil : L'importance de l'étape de modélisation conceptuelle

Étant 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 ambiguë. 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 étapes 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.

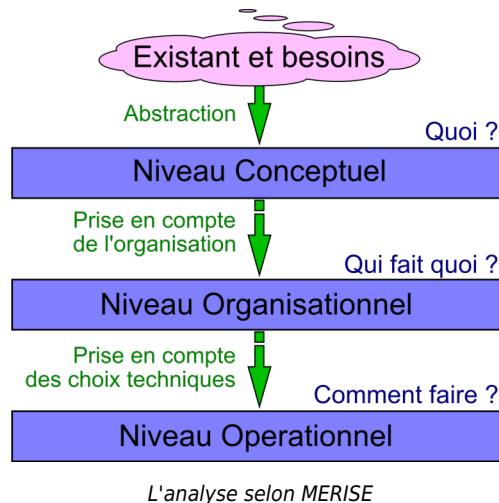
Remarque : Les étapes de traduction logique et d'implémentation

Des logiciels spécialisés (Sybase PowerDesigner [w_sybase] pour la modélisation E-A ou Objecteering software [w_objecteering] 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 modèles logiques (normalisation), puis les implémentations en fonction d'un contexte de mise en oeuvre matériel, logiciel et humain.

1.3.2 La méthode MERISE et le modèle E-A

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



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 ambiguë et donc un bon outil de spécification détaillée

1.3.3 Le langage de modélisation 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® et en tant que outil très utilisé pour la programmation orientée objet, il est amené à supplanter petit à petit la modélisation E-A.

1.3.4 Éléments 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.



Méthode : 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.



Exemple : Exemples de document existants à analyser

- 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.



Méthode : 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.



Exemple : Exemples d'experts à consulter

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

- etc.



Méthode : 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.).



Exemple : Exemples d'utilisateurs avec qui dialoguer

- Personnes qui vont effectuer les saisies d'information (à partir de quelles sources ? Quelle est leur responsabilité ? 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.



Méthode : 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 travaillant 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.



Exemple : Exemples d'autres systèmes coexistants à étudier

- Autres bases de données (les données sont-elles 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.

1.3.5 Le MCD



Définition : MCD

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



Exemple

Le modèle E-A est un formalisme de MCD, le diagramme de classe UML en est un autre.



Remarque

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

1.3.6 Le MLD

Introduction

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 : SGFI (qui ne sont pas vraiment des SGBD), SGBD hiérarchiques (organisés selon une arborescence), SGBD réseau (encore appelés CODASYL), SGBDR, SGBDOO, SGBDRO, etc.



Définition : MLD

Un MLD est une **représentation** du système tel qu'il sera implanté 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 (en SQL avec Oracle par exemple)

1.4 En résumé : Conception de bases de données

Conception

- Modèle Conceptuel
 - Schéma conceptuel canonique et schémas externes
 - Exemples
 - E-A
 - UML
- Modèle Logique
 - Schéma interne indépendant d'un SGBD
 - Exemples
 - Relational
 - Objet
 - Relationnel-Objet
 - Réseau
 - Hiérarchique
- Modèle Physique
 - Schéma interne pour un SGBD particulier
 - Exemples
 - Oracle
 - MySQL
 - PostgreSQL
 - DB2
 - Access
 - SQLServer

* *

*

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

Le niveau conceptuel : la modélisation des bases de données

2

La **modélisation** est l'étape **fondatrice** du processus 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. Après avoir rappelé succinctement les fondements et objectifs des SGBD, ce chapitre proposera les outils méthodologiques nécessaires à la modélisation, à travers les formalismes E-A et UML.

2.1 Bases du diagramme de classes UML

Objectifs

Savoir faire un modèle conceptuel.

Savoir interpréter un modèle conceptuel.

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 qu'une introduction au diagramme de classes (parmi l'ensemble des outils d'UML), limité aux aspects particulièrement utilisés en modélisation de bases de données.

2.1.1 Présentation d'UML

UML est un langage de représentation destiné en particulier à la modélisation objet. UML est devenu une norme OMG en 1997. 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.

- Un langage de représentation destiné en particulier à la modélisation objet
- Une norme OMG en 1997.
- Un formalisme qui impose de "penser objet", indépendant d'un langage de programmation donné.
- Une notation graphique
- 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).

2.1.2 Classes

 Définition : 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()

Représentation UML d'une classe



Exemple : La classe Voiture

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

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.

2.1.3 Attributs



Définition : Attribut

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

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).
- **Un attribut peut être composé** (ou composite) : Il joue alors le rôle d'un groupe d'attributs (par exemple une adresse peut être un attribut composé des attributs numéro, type de voie, nom de la voie). Cette notion renvoie à la notion de variable de type Record dans les langages de programmation classiques.



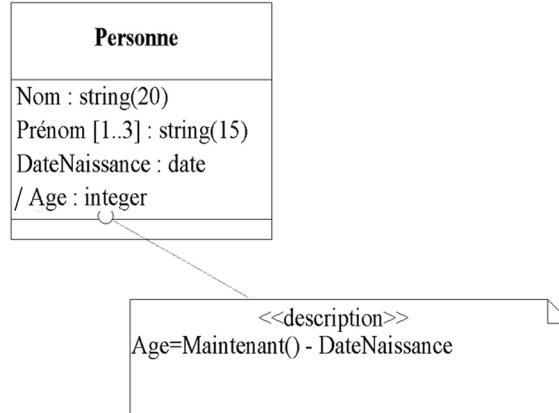
Syntaxe

attribut:type

```

attribut_multivalué[nbMinValeurs..nbMaxValeurs]:type
/attribut_dérivé:type
attribut_composé
- sous-attribut1:type
- sous-attribut2:type
- ...
  
```

Exemple : La classe Personne



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.

Conseil : Attribut composé multivalué

Un attribut peut être composé et multivalué. On préfère néanmoins dans ce cas l'utilisation de l'écriture d'une **composition** équivalente.

Rappel : Voir aussi

Composition (cf. Composition p 24)

2.1.4 Méthodes

Définition : Méthode

Une méthode (ou opération) est une fonction associée à une classe d'objet qui permet d'agir sur les objets de la classe ou qui permet à ces objets de renvoyer des valeurs (calculées en fonction de paramètres).

Syntaxe

```
methode(paramètres):type
```

Remarque : Méthodes et modélisation de BD

Pour la modélisation des bases de données, les méthodes sont surtout utilisées pour représenter des données calculées (à l'instar des attributs dérivés) ou pour mettre en exergue des fonctions importantes du système cible. Seules les méthodes les plus importantes sont représentées, l'approche est moins systématique qu'en modélisation objet par exemple.

Remarque : Méthodes, relationnel, relationnel-objet

Lors de la transformation du modèle conceptuel UML en modèle logique relationnel, les méthodes **ne seront pas implémentées**. Leur repérage au niveau conceptuel sert donc surtout d'aide-mémoire pour l'implémentation au niveau applicatif.

Au contraire, un modèle logique relationnel-objet **permettra l'implémentation** de méthodes associées à des tables. Leur repérage au niveau conceptuel est donc encore plus important.

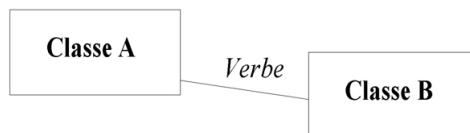
2.1.5 Associations

Définition : 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



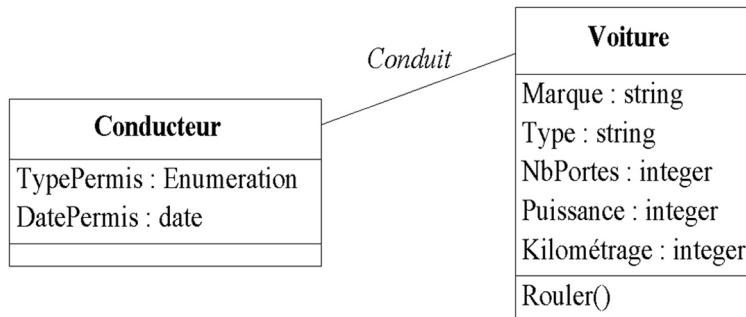
Notation de l'association en UML

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



Représentation d'association en UML

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

Complément: Voir aussi



- Cardinalité (cf. Cardinalité p 19)
- Explication des associations (cf. Explication des associations p 22)
- Associations ternaires (cf. Associations ternaires p 23)
- Contraintes sur les associations (cf. Contraintes sur les associations p 28)

2.1.6 Cardinalité



Définition : 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 adoptée 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.



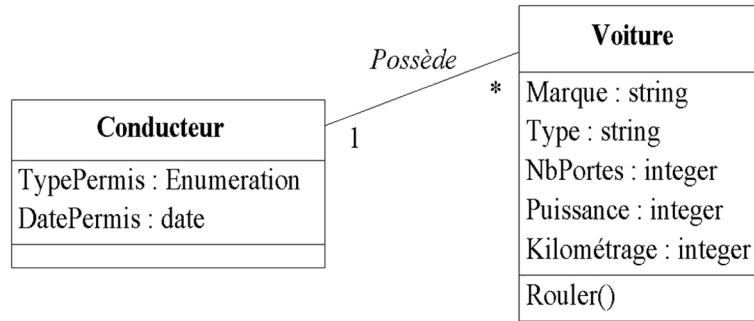
Remarque

Les cardinalités 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



Représentation de cardinalité en UML

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

2.1.7 Héritage



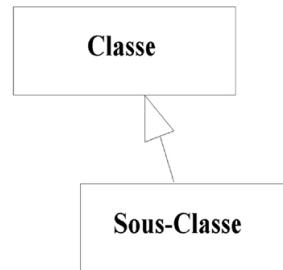
Définition : Héritage

L'héritage est l'association 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



Notation de l'héritage en UML



Remarque : Is-a

L'héritage permet de représenter la relation "est-un" entre deux objets (*is-a* en anglais).

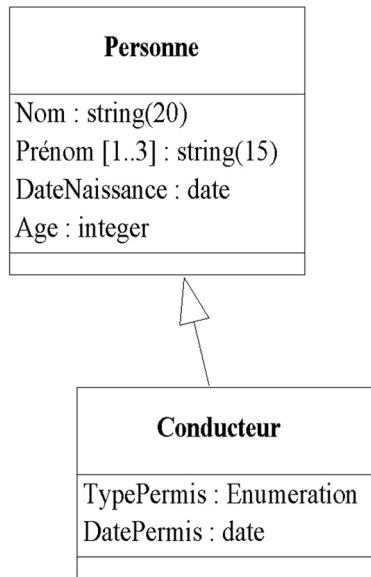


Remarque : Factorisation

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



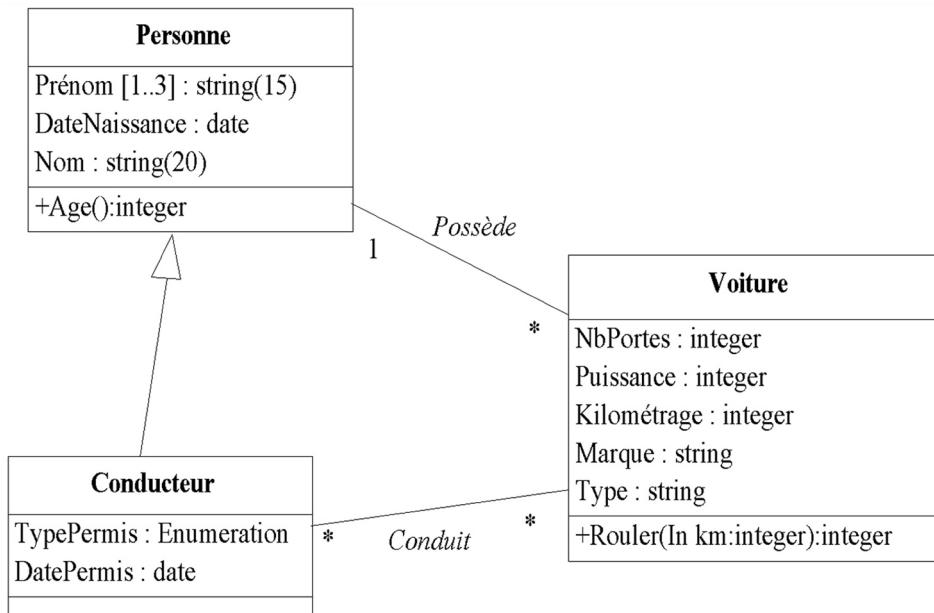
Représentation d'héritage en UML

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.

2.1.8 Exemple : Des voitures et des conducteurs



Exemple



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.



Attention

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.

2.2 Diagramme de classes UML avancé

Objectifs

Maîtriser le diagramme de classe UML dans le cas de la conception de BD.

Les éléments de modélisation suivant complètent les notations basiques du diagramme de classe : classe, attribut, association, cardinalité et héritage.

2.2.1 Explication des associations



Syntaxe : Sens de lecture

Il est possible d'ajouter le sens de lecture du verbe caractérisant l'association sur un diagramme de classe UML, afin d'en faciliter la lecture. On ajoute pour cela un signe < ou > (ou un triangle noir) à côté du nom de l'association

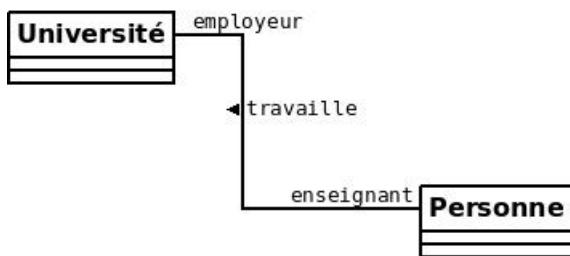


Syntaxe : Rôle

Il est possible de préciser le rôle joué par une ou plusieurs des classes composant une association afin d'en faciliter la compréhension. On ajoute pour cela ce rôle à côté de la classe concernée (parfois dans un petit encadré collé au trait de l'association).



Exemple



Rôle et sens de lecture sur une association



Remarque : Associations réflexives

L'explicitation des associations est particulièrement utile dans le cas des associations réflexives.

2.2.2 Classe d'association

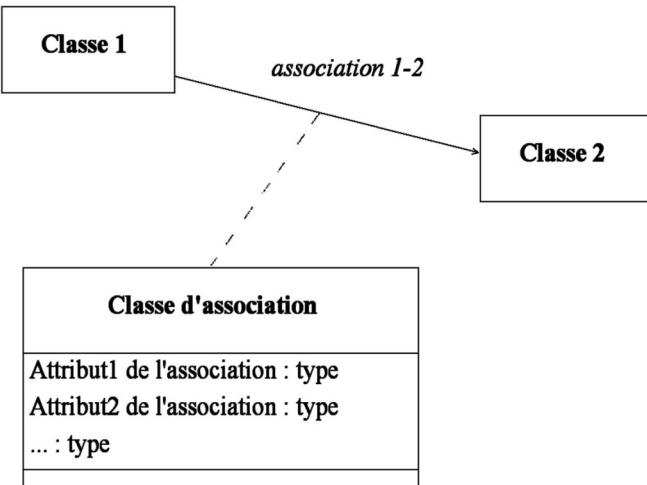


Définition : Classe d'association

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



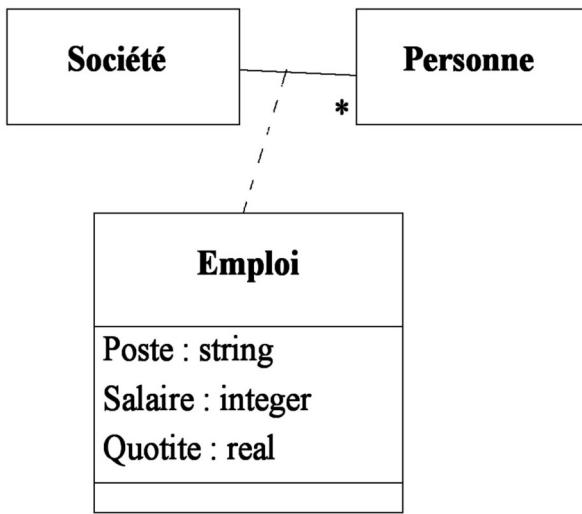
Syntaxe : Notation d'une classe d'association en UML



Notation d'une classe d'association en UML



Exemple : Exemple de classe d'association

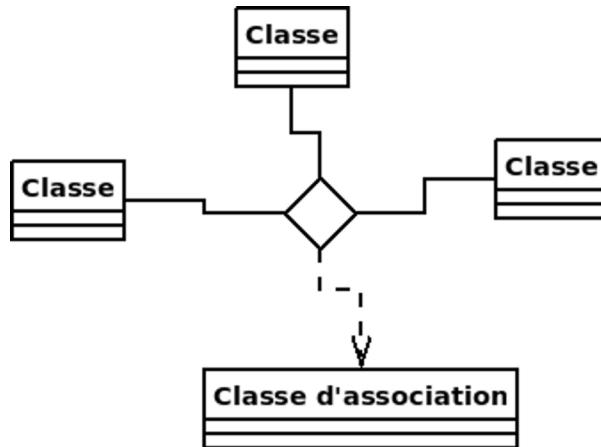


Emplois

2.2.3 Associations ternaires



Syntaxe



Notation d'une association ternaire



Conseil : Ne pas abuser des associations ternaires

Il est toujours possible de réécrire une association ternaire avec trois associations binaires, en transformant l'association en classe.



Conseil : Pas de degré supérieur à 3

En pratique on n'utilise jamais en UML d'association de degré supérieur à 3.

2.2.4 Composition



Définition : 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 (voire 1:1).
- 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 disparaît.

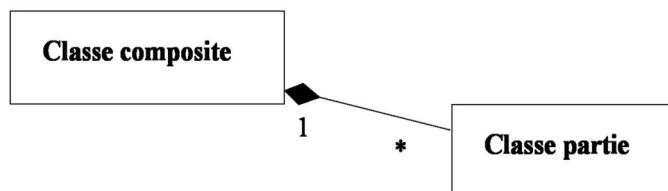


Remarque

- La composition est une association particulière (binnaire de cardinalité contrainte).
- La composition n'est pas symétrique, une classe joue le rôle de conteneur pour les classes liées, elle prend donc un rôle particulier a priori.
- La composition est une agrégation avec des contraintes supplémentaires (non partageabilité et cycle de vie lié).



Syntaxe : Notation d'une composition en UML



Notation de la composition en UML



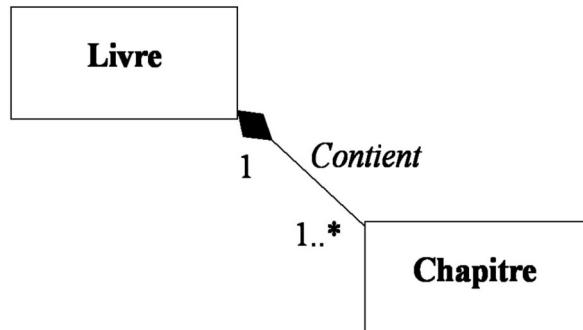
Attention: Composition et cardinalité

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 : Exemple de composition



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-AI. L'entité de type faible correspond à un objet partie et l'entité identifiante à un objet composite.



Conseil : Composition et attribut multivalué

- Une composition avec une classe partie dotée d'un seul attribut peut s'écrire avec un attribut multivalué.
- Un attribut composé et multivalué peut s'écrire avec une composition.



Rappel : Voir aussi

- *Attribut composé multivalué* (cf. Attributs p 17)
- *Agrégation* (cf. Agrégation p 25)

2.2.5 Agrégation



Définition : Association d'agrégation

L'agrégation est une association particulière utilisée pour préciser une relation tout/partie (ou ensemble/élément), on parle d'**association méréologique**.

Elle possède la propriété suivante : L'agrégation associe une classe agrégat et des classes parties, tel que tout objet partie appartient à au moins un objet agrégat.

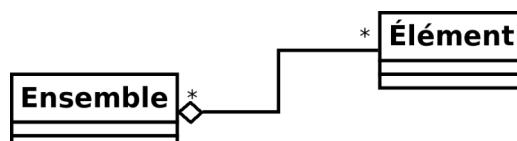


Remarque

- L'agrégation est une association particulière (binnaire de cardinalité libre).
- L'agrégation n'est pas symétrique.



Syntaxe



Notation de l'agrégation en UML

La cardinalité, peut être exprimée librement, en particulier les instances de la classe Élément peuvent être associées à plusieurs instances de la classe Ensemble, et même de plusieurs classes.



Attention

L'agrégation garde toutes les propriétés d'une association classique (cardinalité, cycle de vie, etc.), elle ajoute simplement une terminologie un peu plus précise via la notion de tout/partie.

2.2.6 Classes abstraites



Définition : 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.



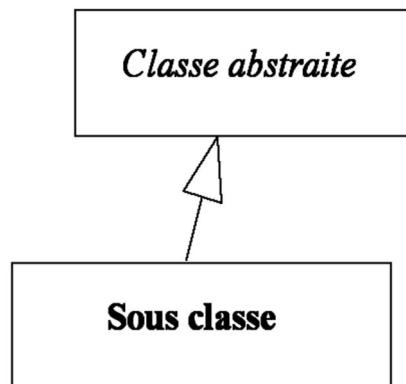
Attention: Classe abstraite et héritage

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 ait un sens.



Syntaxe : Notation d'une classe abstraite en UML

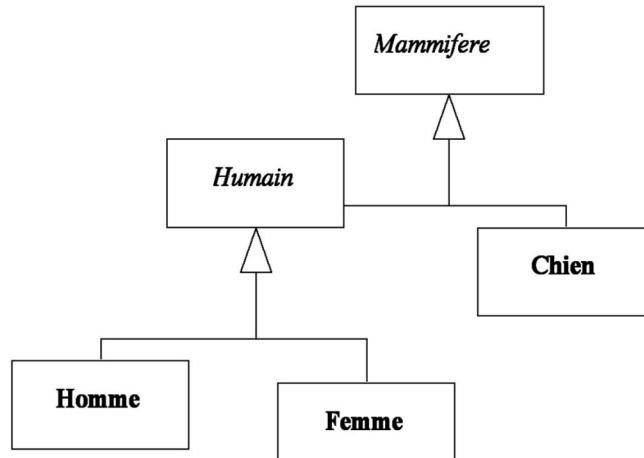
On note les classes abstraites en italique.



Notation d'une classe abstraite en UML



Exemple : Exemple de classes abstraites



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.

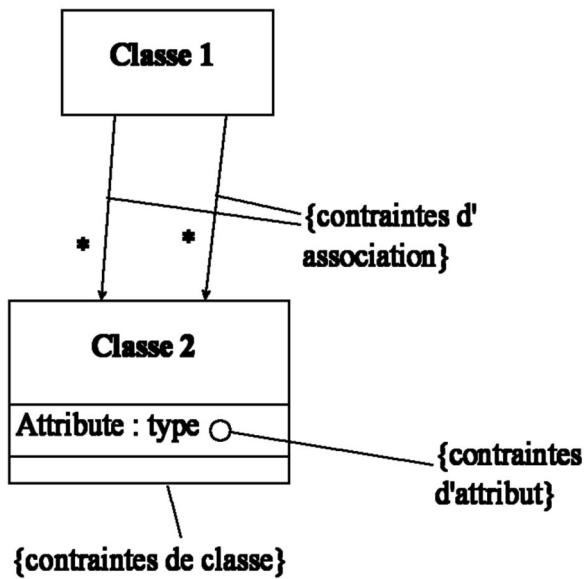
2.2.7 Contraintes

Ajout de contraintes dynamiques sur le diagramme de classe

Il est possible en UML d'exprimer des contraintes dynamiques sur le diagramme de classe, par annotation de ce dernier.



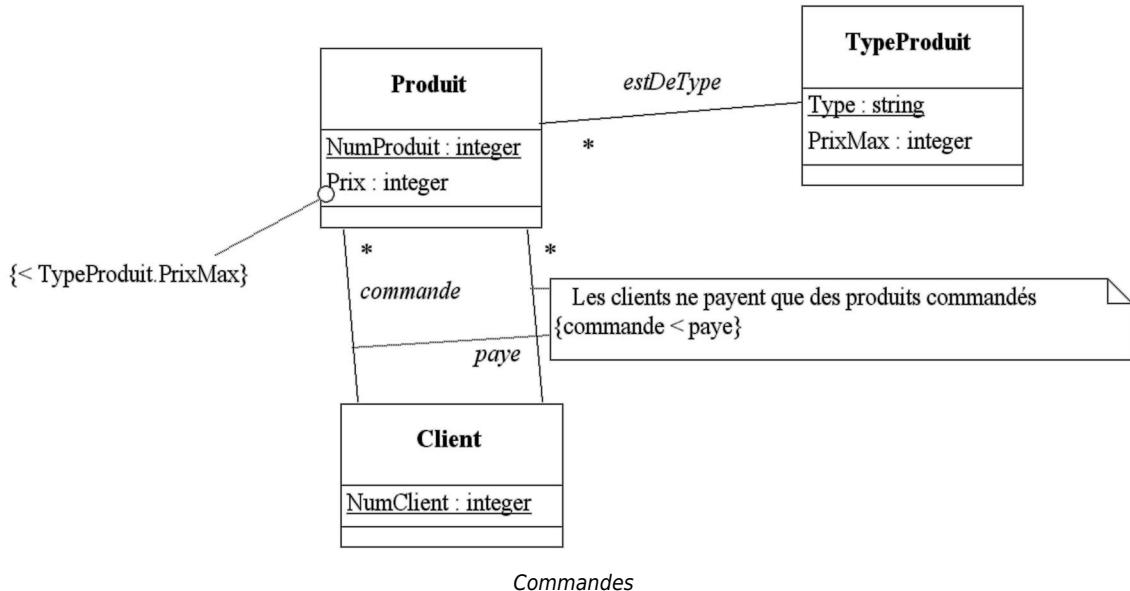
Syntaxe : Notation de contraintes



Notation contraintes en UML



Exemple : Exemple de contraintes



Commandes



Méthode : Expressions formelles ou texte libre ?

Il est possible d'exprimer les contraintes en texte libre ou bien en utilisant des expressions formelles.

On privilégiera la solution qui offre le meilleur compromis entre facilité d'interprétation et non ambiguïté. La combinaison des deux est également possible si nécessaire.



Méthode : Quelles contraintes exprimer ?

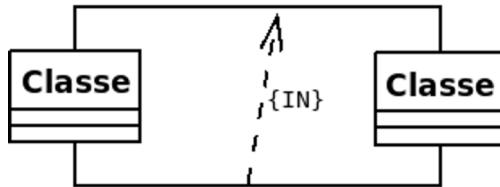
En pratique il existe souvent de très nombreuses contraintes, dont certaines sont évidentes, ou encore secondaires. Or l'expression de toutes ces contraintes sur un diagramme UML conduirait à diminuer considérablement la lisibilité du schéma sans apporter d'information nécessaire à la compréhension. En conséquence on ne représentera sur le diagramme que les contraintes les plus essentielles.

Notons que lors de l'implémentation toutes les contraintes devront bien entendu être exprimées. Si l'on souhaite préparer ce travail, il est conseillé, lors de la modélisation logique, de recenser de façon exhaustive dans un tableau toutes les contraintes à implémenter.

2.2.8 Contraintes sur les associations

Concernant les associations, il existe une liste de contraintes exprimées de façon standard.

Syntaxe



Notation des contraintes sur les associations

Définition : Inclusion

Si l'association inclue est instanciée, l'autre doit l'être aussi (la contrainte d'inclusion a un sens, représenté par une flèche).

Syntaxe

{IN}, également notée {Subset} ou {!}.

Définition : Et (égalité, simultanéité)

Si une association est instanciée, l'autre doit l'être aussi (équivalent à une double inclusion).

Syntaxe

{AND}, également notée {=} ou {S} pour simultanéité.

Définition : Exclusion

Les deux associations ne peuvent être instanciés en même temps.

Syntaxe

{X}

Définition : Ou inclusif (couverture, totalité)

Au moins une des deux relations doit être instanciée.

Syntaxe

{OR}, également noté {T} pour totalité.

Définition : Ou exclusif (partition)

Exactement une des deux relations doit être instanciée.

Syntaxe

{XOR}, également notée {+} ou {XT} ou {Partition}.

2.2.9 Paquetages

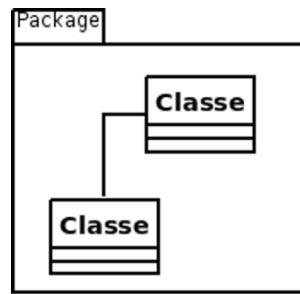
Définition : Package

Les paquetages (plus communément appelés package) sont des éléments servant à organiser un modèle.

Ils sont particulièrement utiles dès que le modèle comporte de nombreuses classes et que celles-ci peuvent être triées selon plusieurs aspects structurants.



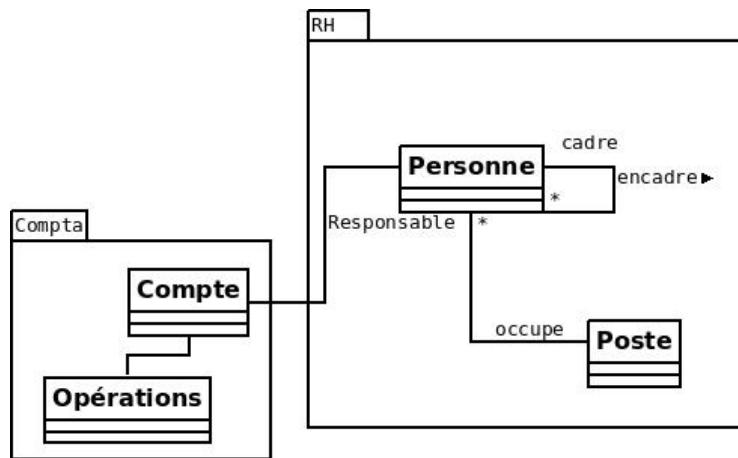
Syntaxe



Notation des paquetages en UML



Exemple



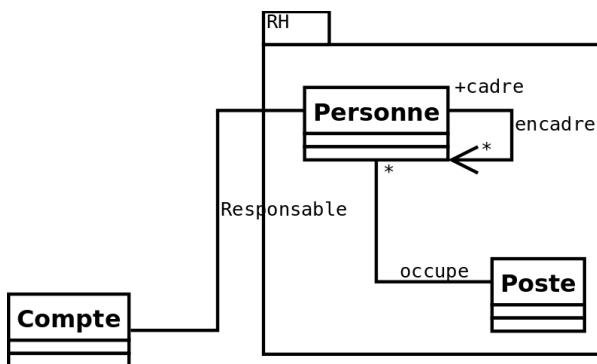
Exemple d'utilisation des packages



Méthode

On représente chaque classe au sein d'un *package*. Il est alors possible de faire une présentation globale du modèle (tous les *packages*), partielle (une partie des *packages*) ou centrée sur un seul *package*.

Pour une représentation partielle ou centrée sur un *package*, on représente les *packages* concernés avec leurs classes propres, ainsi que toutes les classes liées des autres packages (et seulement celles-ci).



Présentation partielle du modèle centrée sur un package

2.2.10 Stéréotype



Définition : Stéréotype UML

Un stéréotype UML est une syntaxe permettant d'ajouter de la sémantique à la modélisation des classes. Il permet de définir des **types de classe**, afin de regrouper conceptuellement un ensemble de classes (à l'instar d'une classe qui permet de regrouper conceptuellement un ensemble d'objets).

C'est une mécanique de métamodélisation : elle permet d'étendre le métamodèle UML, c'est à dire le modèle conceptuel du modèle conceptuel.



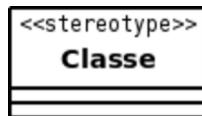
Définition : Méta-modèle

Un méta-modèle est le modèle d'un modèle. Par exemple le méta-modèle UML comprend les concepts de classe, attribut, association, cardinalité, composition, agrégation, contraintes, annotations, ... On mobilise ces concepts (on les instancie) pour exprimer un modèle particulier suivant le formalisme UML.

Les stéréotypes permettent donc d'ajouter au méta-modèle UML standard, celui que tout le monde utilise, des concepts locaux pour enrichir le langage de modélisation que l'on utilise pour réaliser des modèles.



Syntaxe



Notation d'un stéréotype en UML



Conseil : Stéréotypes spécifiques et stéréotypes standard

Un stéréotype spécifique enrichit le méta-modèle UML, mais selon une sémantique qui est propre à celui qui l'a posé, non standard donc. La conséquence est que pour un tiers, l'interprétation du stéréotype n'est plus normalisée, et sera potentiellement plus facilement erronée. Il convient donc de ne pas abuser de cette mécanique.

Deux ou trois stéréotypes spécifiques, correctement définis, sont faciles à transmettre, plusieurs dizaines représenteraient un nouveau langage complet à apprendre pour le lecteur du modèle.

Il existe des stéréotypes fournis en standard par UML, ou communément utilisés par les modélisateurs. L'avantage est qu'il seront compris plus largement, au même titre que le reste du méta-modèle (ils ont une valeur de standard).

2.3 Le modèle E-A

Objectifs

Savoir faire un modèle E-A étendu.

Savoir interpréter un modèle E-A étendu.

2.3.1 Le modèle E-A en bref

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

Il correspond au niveau conceptuel de la méthode MERISE (méthode d'analyse informatique), le MCD (cf. Méthode MERISE Tome 1 : Principes et outils [Tardieu83], Méthode MERISE Tome 2 : Démarche et pratiques [Tardieu85]).

La conception E-A est issue des travaux de Chen [Chen76] et se fonde sur deux concepts principaux et un troisième sous-jacent : l'entité, l'association et l'attribut ou propriété.

2.3.2 Entité



Définition : 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.

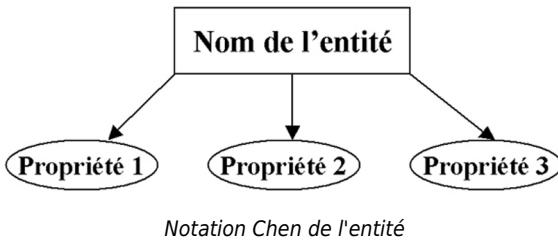
L'occurrence d'une entité est un élément particulier correspondant à l'entité et associé à un élément du réel. Chaque entité a des propriétés (ou attributs) qui la décrivent. Chaque attribut est associé à un domaine de valeur. Une occurrence a des valeurs pour chacun de ses attributs, dans le domaine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



Syntaxe

ENTITE	
ATTRIBUT_CLE	Domaine
ATTRIBUT_1	Domaine
ATTRIBUT_2	Domaine
ATTRIBUT_N	Domaine

Notation MERISE de l'entité



Remarque

- Un attribut est atomique, c'est à dire qu'il ne peut prendre qu'une seule valeur pour une occurrence.
- 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 occurrence est appelé attribut **clé**.

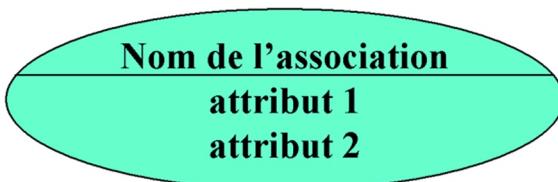
2.3.3 Association

Définition : Association

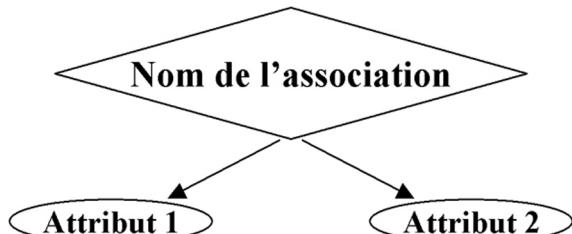
Une association (ou type d'association) représente un lien quelconque entre différentes entités.

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



Notation MERISE de l'association



Notation Chen de l'association

Remarque

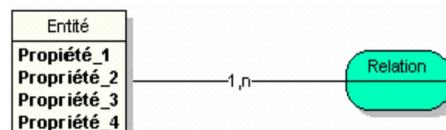
On peut avoir plusieurs associations différentes définies sur les mêmes entités.

2.3.4 Cardinalité d'une association

Définition : 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.

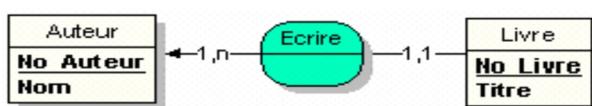
Syntaxe



Notation de la cardinalité



Exemple



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).

2.3.5 Modèle E-A étendu

Introduction

On peut étendre le modèle E-A "classique" de façon à accroître 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.

2.3.5.1 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.).

2.3.5.2 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.

2.3.5.3 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.

2.3.5.4 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.

2.3.6 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".

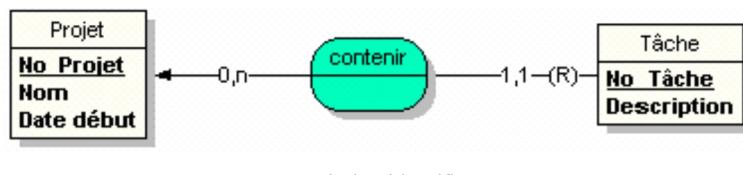


Définition : Entité de type faible

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



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.



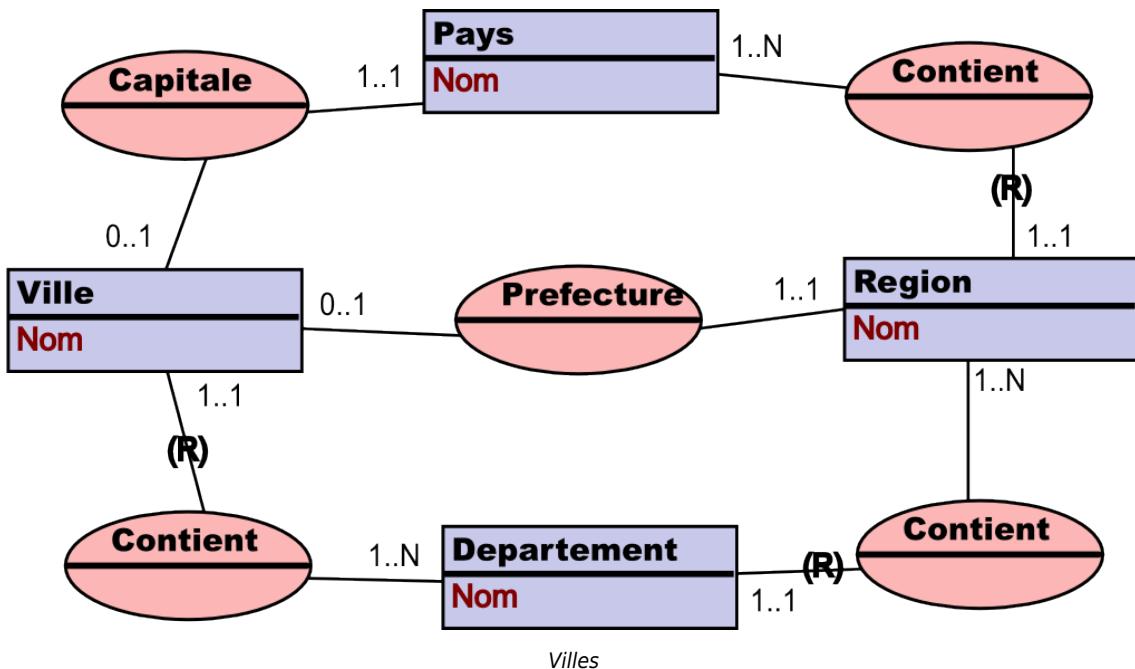
Attention

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.

2.3.7 Illustration d'entités faibles



Exemple



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, concrètement, 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.

2.4 En résumé : Schéma conceptuel

Schéma conceptuel

Un schéma conceptuel peut être représenté sous forme de modèle E-A ou sous forme de diagramme de classe UML.

- Entité ou Classe
 - Propriété ou Attribut
 - Typé
 - Multi-valué
 - Composé
 - Dérivé
 - Méthode
 - Paramètres
 - Valeur de retour
- Association
 - Association
 - Verbe
 - Cardinalité
 - Héritage
 - Héritage d'attributs
 - Héritage de méthodes
 - Composition (ou entité faible)
 - Cardinalité

2.5 Bibliographie commentée sur la modélisation UML



Complément: Outils de modélisation UML

Il existe de nombreux outils de modélisation UML. On pourra citer :

- Dia [w_dia] : logiciel Open Source et multi-plateformes facile d'usage (qui marche néanmoins mieux sur Linux que sur Windows).
- Objecteering [w_objecteering] (version gratuite).

À voir également en Open Source : ArgoUML¹ ou EclipseUML² (non testé par l'auteur).



Complément: Modélisation UML

UML2 en action [Roques04]

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'agrégation
- 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)

1 - <http://argouml.tigris.org/>

2 - <http://www.eclipsedownload.com/>

Le niveau conceptuel : la modélisation des bases de données

- Des principes d'introduction de métaclasses (type)s



Complément: Référence UML en ligne

UML en Français [w.uml.free.fr]

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>



Complément: Tutoriel sur la modélisation UML.

UML en 5 étapes [[w.journaldunet.com\(1\)](http://w.journaldunet.com/1)]

On consultera en particulier le tutoriel sur les diagrammes de classe :

http://developpeur.journaldunet.com/tutoriel/cpt/010607cpt_umlintro.shtml



Complément: Conseils

Cinq petits conseils pour un schéma UML efficace [[w.journaldunet.com\(2\)](http://w.journaldunet.com/2)]



Complément: Pratique

UML2 par la pratique [Roques09] (chapitre 3)

Des explications, exemples et études de cas.

Le modèle relationnel est aux fondements des SGBDR. Il a été - et continue d'être - le modèle théorique dominant pour la représentation logique des BD. Le modèle relationnel permet de reformuler le modèle conceptuel dans un formalisme beaucoup plus proche de l'implémentation informatique, bien que encore indépendant d'une solution technologique particulière.

Le modèle relationnel, et en particulier l'algèbre relationnelle qui lui est associée, est aussi le fondement théorique du langage standard SQL, qui est utilisé pour manipuler les données stockées dans une BD.

3.1 Description du modèle relationnel

Objectifs

Connaître les fondements théoriques du modèle relationnel.

Comprendre le principe d'éclatement des relations et de non-redondance.

3.1.1 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 [Codd70] s'est imposé 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.

3.1.2 Le modèle relationnel

Introduction

Le modèle relationnel a été introduit par Codd [Codd70], en 1970 au laboratoire de recherche d'IBM de San José. Il s'agit d'un modèle simple et puissant à la base de la majorité des bases de données aujourd'hui.

 Définition : 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.

3.1.3 Domaine



Définition : Domaine

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



Remarque

Un domaine peut-être défini en intension (c'est à dire en définissant les propriétés caractéristiques 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}

3.1.4 Produit cartésien



Définition : 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) $\langle V_1, V_2, \dots, V_n \rangle$ tel que V_i est une valeur de D_i 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>, }
```

3.1.5 Relation



Définition : 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.

Synonymes : Table, tableau



Syntaxe

On peut représenter la relation R sur les domaines 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

Tableau 1 Relation R



Remarque

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



3.1.6 Attribut et enregistrement

Définition : 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

Définition : 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

Tableau 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".

3.1.7 Exemple : 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

Tableau 3 Relation Vol

3.1.8 Clé

 Définition : Clé

Une clé est un groupe d'attributs minimum qui détermine un tuple unique dans une relation.

Toute relation doit comporter au moins une clé, ce qui implique qu'une relation ne peut contenir deux tuples identiques.

 Attention: 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.

 Définition : Clé primaire

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).

 Définition : Clés candidates

On appelle **clés candidates** l'ensemble des clés d'une relation qui n'ont pas été choisies comme clé primaire (elles étaient candidates à cette fonction).

 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.

3.1.9 Clé artificielle



Définition : 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 un attribut supplémentaire ajouté au schéma de la relation, qui n'est lié à aucune signification, et qui sert uniquement à identifier de façon unique les enregistrements et/ou à simplifier les références de clés étrangères.



Définition : Clé signifiante

Une clé est signifiante si elle n'est pas artificielle.



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 répercussion 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'en 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 informations 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.

3.1.10 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_n, 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.

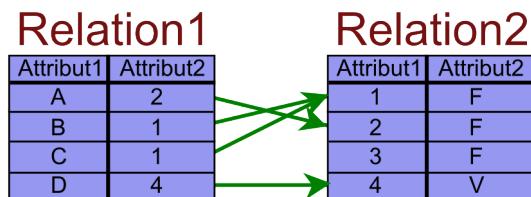


Méthode : Lien

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



Principe des liens entre relations

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.

3.1.11 Éclatement des relations pour éviter la redondance

Relation redondante

NumTrajet	Date	Gare1	Gare2	Train	Vitesse	NumSec	Nom	Prénom
1010	01-01-2001	Paris	Lyon	TGV	450	2750175001	Dupont	Joëlle
1011	02-01-2001	Paris	Limoges	TER	200	1700275002	Durand	Jean-Pierre
1012	03-01-2001	Paris	Madrid	TGV	450	2750175001	Dupont	Joëlle
1013	03-01-2001	Lyon	Limoges	TER	200	1700275002	Dupont	Jean-Pierre

Tableau 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'attributs 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 disparaîtront. 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 préférera donc un découpage de la relation VoyageEnTrain en trois relations : Voyage, Modele et Conducteur, chacune représentant des informations correspondant à des objets différents.

NumTrajet	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

Tableau 5 Relation Voyage

Modele	Vitesse
TGV	450
TER	200

Tableau 6 Relation Modele

NumSec	Nom	Prénom
2750175001	Dupont	Joëlle
1700275002	Durand	Jean-Pierre

Tableau 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	2750175001
1011	02-01-2001	Paris	Limoges	TER	1700275002
1012	03-01-2001	Paris	Madrid	TGV	2750175001
1013	03-01-2001	Lyon	Limoges	TER	1700275002

Tableau 8 Relation Voyage

3.1.12 Clé étrangère

 Définition : 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.

 Définition : 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 référencants 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 suppression en cascade (sachant que les tuples référencant 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.

3.1.13 Schéma relationnel



Définition : Schéma d'une relation

Le schéma d'une relation définit cette relation par intension. 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.



Définition : Schéma relationnel d'une base de données

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" (attribut précédés de # ou bien soulignés)

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 réfèrent quels attributs de la clé primaire (cela est généralement évident) mais il est possible de la faire en 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).



Syntaxe : Clé candidates

```
Relation1 (...AttributM:DomaineM, ....) avec AttributM clé
```

Les clés candidates doivent être notées sur le schéma relationnel :

- S'il n'y a qu'une ou deux clés candidates, les noter directement après la définition de la relation
- S'il y a beaucoup de clés, pour ne pas trop alourdir la notation, renvoyer à un tableau à part



Attention: Clés candidates et clé primaire

La notation R(#a,#b) signifie toujours que R a comme clé primaire (a,b), et non que R aurait deux clés a et b (dont on ne saurait pas laquelle est primaire).

La notation R(#a,b) avec b clé signifie bien que a et b sont deux clés de R, et que a est primaire.

Il ne faut pas confondre **une clé composée de deux attributs** avec **deux clés**.

3.1.14 Exemple de schéma relationnel pour la géographie



Exemple

```
Personne (#Numéro: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 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 identifiées par un code postal qui est unique dans le monde (en utilisant le préfixe 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.

3.2 Passage UML-Relationnel : Cas systématiques

Objectifs

Savoir faire le passage d'un schéma conceptuel UML à un schéma relationnel dans les cas simples.

Reconnaître les cas de transformation qui se traitent toujours de la même façon.

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 que ce passage soit systématisé dans la plupart des cas.

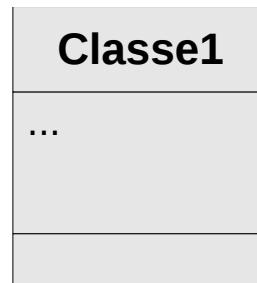
3.2.1 Transformation des classes



Méthode : Classe

Pour chaque classe non abstraite,

- on crée une relation dont le schéma est celui de la classe ;
- la clé primaire de cette relation est une des clés de la classe.



Graphique 2 Classe

Classe1(...)



Remarque

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

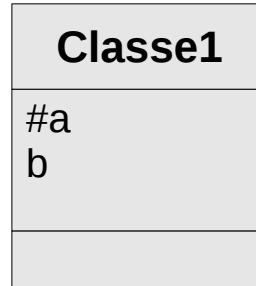
3.2.2 Transformation des attributs



Méthode : Attributs simples

Pour chaque attribut élémentaire et monovalué d'une classe,

- on crée un attribut correspondant.



Graphique 3 Attribut

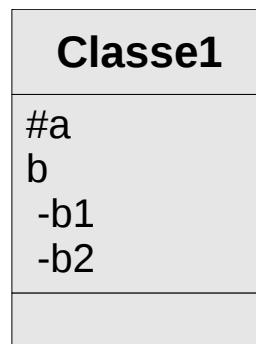
Classe1(#a, b)



Méthode : Attributs composites

Pour chaque attribut composite comprenant N sous-attributs d'une classe,

- on crée N attributs correspondants,
- dont les noms sont la concaténation du nom de l'attribut composite avec celui du sous-attribut.



Graphique 4 Attribut composé

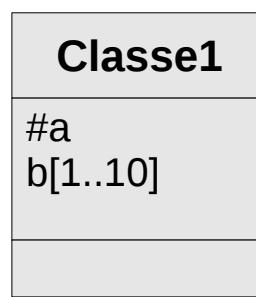
Classe1(#a, b_b1, b_b2)



Méthode : Attributs multivalués

Pour chaque attribut multivalué b d'une classe C,

- on crée une nouvelle relation RB,
- qui comprend un attribut monovalué correspondant à b,
- plus la clé de la relation représentant C ;
- la clé de RB est la concaténation des deux attributs.



Graphique 5 Attribut multivalué

Classe1(#a)

RB(#b, #a=>Classe1)



Méthode : Attributs multivalués (méthode alternative)

Dans le cas où le nombre maximum de b est fini, et petit, on peut également adopter la transformation suivante :

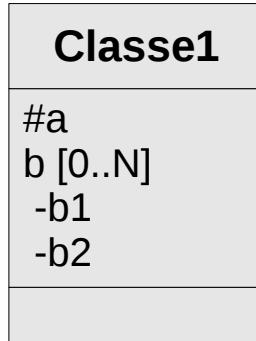
Classe1(#a, b1, b2, b3, b4, b5, b6, b7, b8, b9, b10).

Si le nombre d'attributs est infini ($b[1..*]$) c'est impossible, s'il est trop grand ce n'est pas souhaitable.



Méthode : Attributs composés multivalués

On combine les règles énoncées pour les attributs composés et pour les attributs multivalués.



Graphique 6 Attribut composé multivalué

Classe1(#a)

RB(#b_b1, #b_b2, #a=>Classe1)



Rappel : Voir aussi

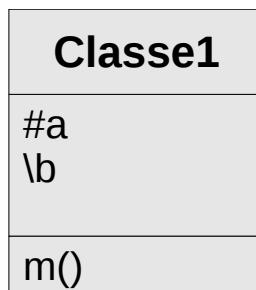
Transformation des compositions (cf. Transformation des compositions p 49)

3.2.3 Transformation des attributs dérivés et méthodes



Méthode : 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.



Graphique 7 Attribut dérivé et méthodes

Classe1(#a)



Remarque : Attribut dérivé stockés

On peut 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 général déconseillé, et ce qui doit être dans tous les cas contrôlé.

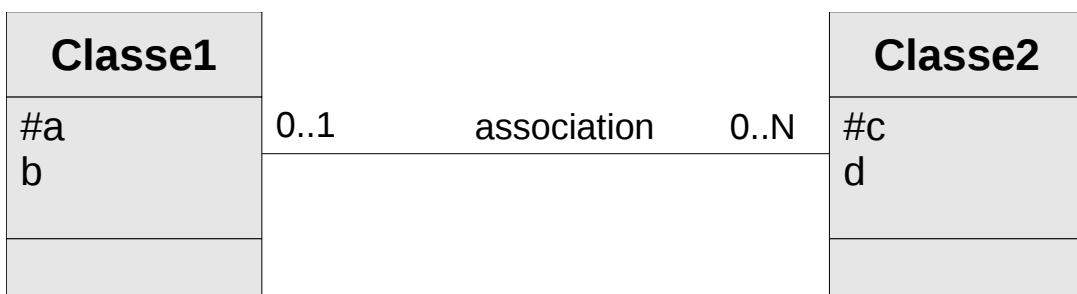
3.2.4 Transformation des associations 1:N



Méthode : Association 1:N

Pour chaque association binaire de type 1:N (ou 0,1:N),

- on ajoute à la relation côté N une clé étrangère vers la relation côté 1.

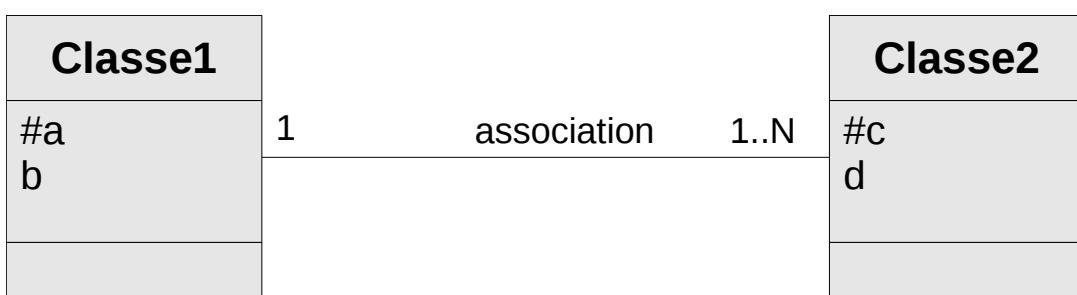


Graphique 8 Association 1:N

Classe1(#a, b)
Classe2(#c, d, a=>Classe1)

Méthode : Cardinalité minimale 1

- Si la cardinalité est exactement 1 (1..1) côté 1, alors on ajoutera une contrainte de non nullité sur la clé étrangère,
- si la cardinalité est au moins 1 (1..N) côté N, on ajoutera une contrainte d'existence de tuples référencant pour chaque tuple de la relation référencée.



Graphique 9 Association 1:N

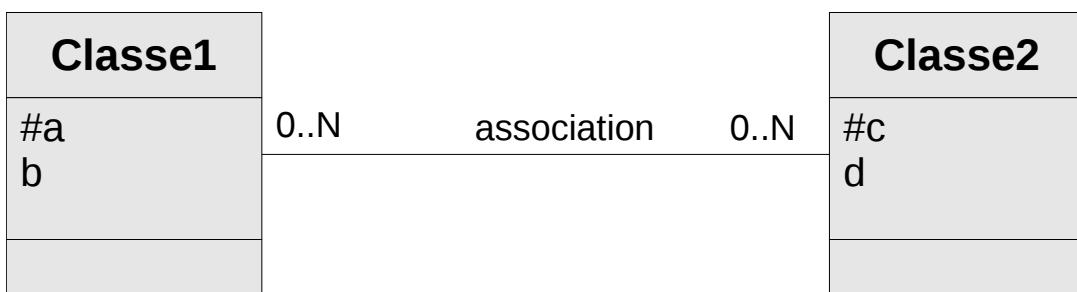
Classe1(#a, b) avec Classe1 IN Classe2
Classe2(#c, d, a=>Classe1) avec a NOT NULL

3.2.5 Transformation des associations N:M

Méthode : Association M:N et associations de degré supérieur à 2

Pour chaque association binaire de type M:N ou de degré supérieur à 2,

- on crée une nouvelle relation,
- composée de clés étrangères vers chaque relation associée,
- et dont la clé primaire est la concaténation de ces clés étrangères.

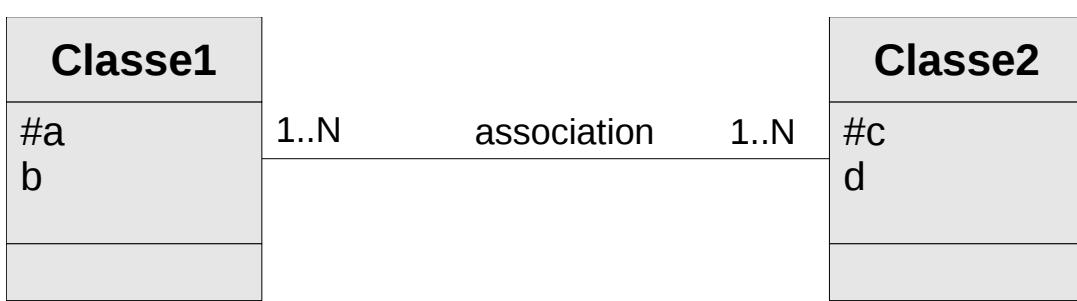


Graphique 10 Association N:M

Classe1(#a, b)
Classe2(#c, d)
Assoc(#a=>Classe1, #c=>Classe2)

Méthode : Cardinalité minimale 1

- Si la cardinalité est exactement au moins 1 (1..N) d'un côté et/ou de l'autre, alors des contraintes d'existence simultanée de tuple devront être ajoutée.
- Ce n'est pas nécessaire si la cardinalité est 0..N.



Graphique 11 Association N:M

Classe1(#a, b) avec Classe1 IN Classe2

Classe2(#c, d) avec Classe2 IN Classe1

Assoc(#a=>Classe1, #c=>Classe2)

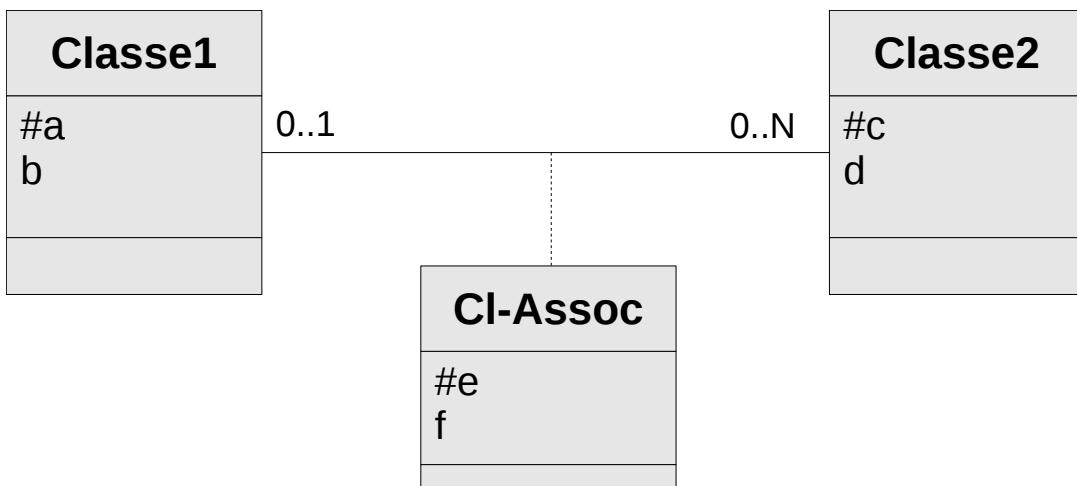
3.2.6 Transformation des classes d'association



Méthode : Classe d'association 1:N

Les attributs de la classe d'association,

- sont ajoutés à la relation issue de la classe côté N ;
- les clés deviennent des clés candidates.



Graphique 12 Classe d'association (1:N)

Classe1(#a, b)

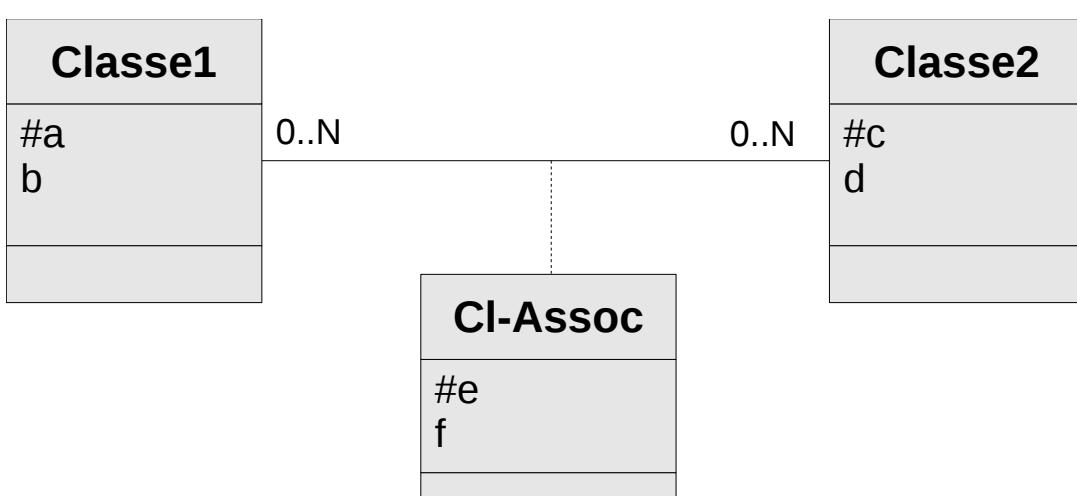
Classe2(#c, d, a=>Classe1, e, f) avec e clé candidate



Méthode : 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 (dite clé locale), celle-ci est concaténée aux clés étrangères composant déjà la clé primaire de la relation d'association.



Graphique 13 Classe association (N:M)

Classe1(#a, b)
Classe2(#c, d)
Assoc(#a=>Classe1, #c=>Classe2, #e, f)

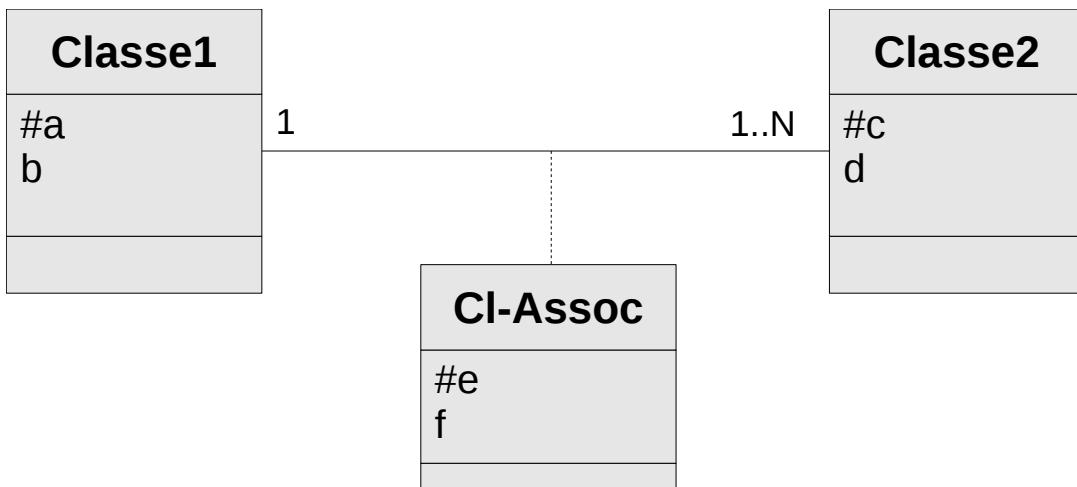


Méthode : Classe d'association 1:1

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



Complément: Cardinalité minimale 1 (1:N)

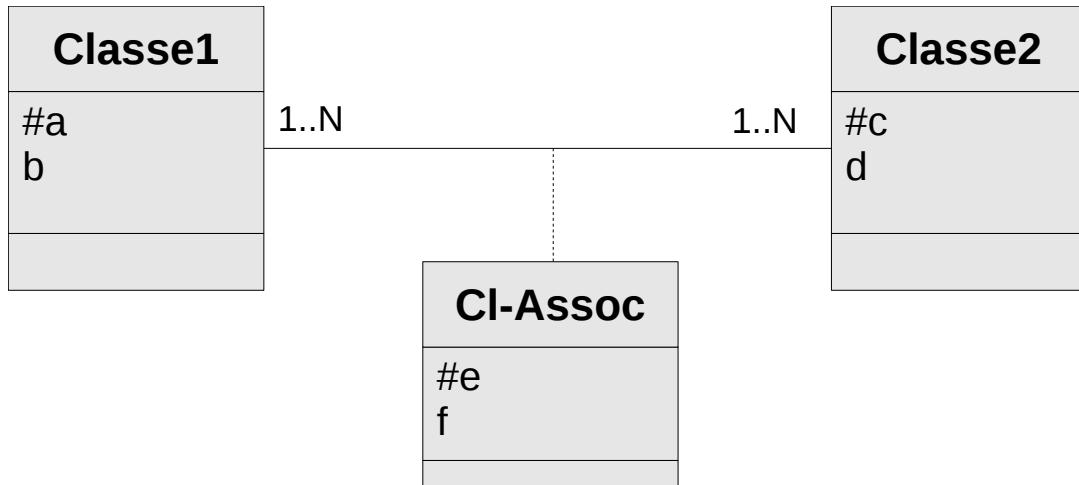


Graphique 14 Classe d'association (1:N)

Classe1(#a, b) avec Classe1 IN Classe2
Classe2(#c, d, a=>Classe1, e, f) avec e clé candidate et a NOT NULL



Complément: Cardinalité minimale 1 (N:M)



Graphique 15 Classe association (N:M)

Classe1(#a,b) avec Classe1 IN Classe2

Classe2(#c,d) avec Classe2 IN Classe1

Assoc(#a=>Classe1,#c=>Classe2,#e,f)

3.2.7 Transformation des agrégations



Rappel : Agrégation

Les associations de type agrégation se traitent de la même façon que les associations classiques.



Graphique 16 Agrégation 1:N

Classe1(#a,b)

Classe2(#c,d,a=>Classe1)



Graphique 17 Agrégation N:M

Classe1(#a,b)

Classe2(#c,d)

Assoc(#a=>Classe1,#c=>Classe2)

3.2.8 Transformation des compositions



Remarque

Une composition

- est transformée comme une association 1:N,
- puis on ajoute à la clé primaire de la classe partie (dite clé locale) la clé étrangère vers la classe composite.



Graphique 18 Composition

Classe1(#a, b)

Classe2(#c, #a=>Classe1, d)



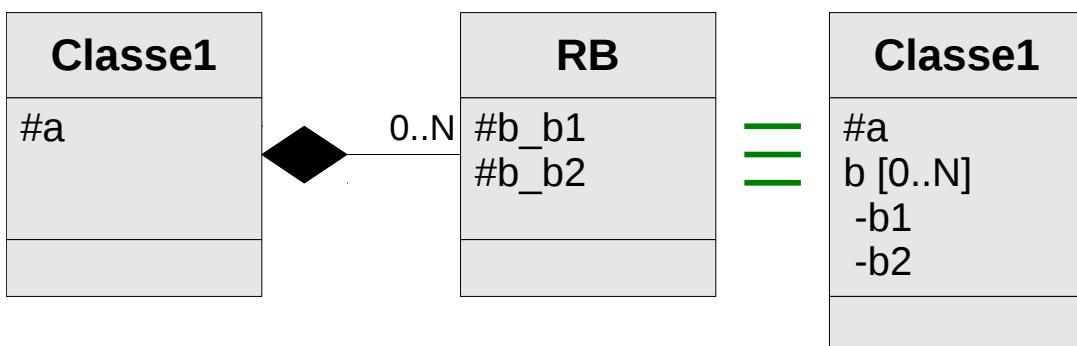
Complément: Composition et entités faibles en E-A

Une composition est transformée selon les mêmes principes qu'une entité faible en E-A.



Complément: Attributs multivalués et composés

La transformation d'une composition donne un résultat équivalent à la transformation d'un attribut composé multivalué.

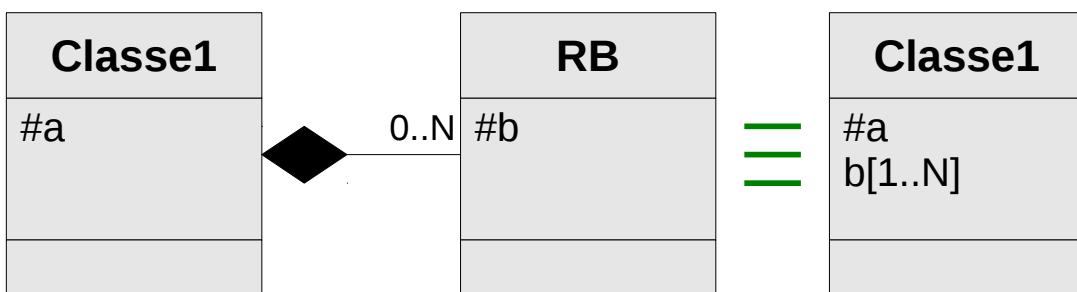


Graphique 19 Composition et attribut composé multivalué

Classe1(#a)

RB(#b_b1, #b_b2, #a=>Classe1)

La transformation d'une composition avec un seul attribut pour la classe composante donne un résultat équivalent à la transformation d'un attribut multivalué.



Graphique 20 Composition et attribut multivalué

Classe1(#a)

RB(#b, #a=>Classe1)



Rappel : Voir aussi

Transformation des attributs (cf. Transformation des attributs p 44)

3.2.9 Liste des contraintes



Méthode : Contraintes exprimées sur le diagramme

Les contraintes exprimées sur les diagrammes sont reportées dans un tableau qui accompagnera le modèle logique. On ajoutera également les clés candidates.

Relation 1	AND (Relation1, Relation4) ; a clé candidate ; ...
Relation 2	b > c ; XOR (Relation2, Relation 3) ; ...

L'on peut également commenter chaque relation directement lorsque les informations ne sont pas trop nombreuses.

Relation1 (a, b, c) avec : c clé candidate ; AND(Relation1, Relation4)
Relation2 (b, c, d) avec : b > c ; XOR(Relation2, Relation3)



Méthode : 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.

3.2.10 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 et clé étrangère
Attribut dérivé	Procédure stockée ou contrainte dynamique
Méthode	Procédure stockée
Association 1:N	Clé étrangère
Association N:M	Relation et clé étrangère
Association de degré 3 ou supérieur	Relation et clé étrangère
Classe d'association	Attributs
Composition	Clé

Tableau 9 Passage UML vers Relationnel

3.3 Passage UML-Relationnel : Associations 1:1

Objectifs

Savoir faire le passage d'un schéma conceptuel UML à un schéma relationnel.

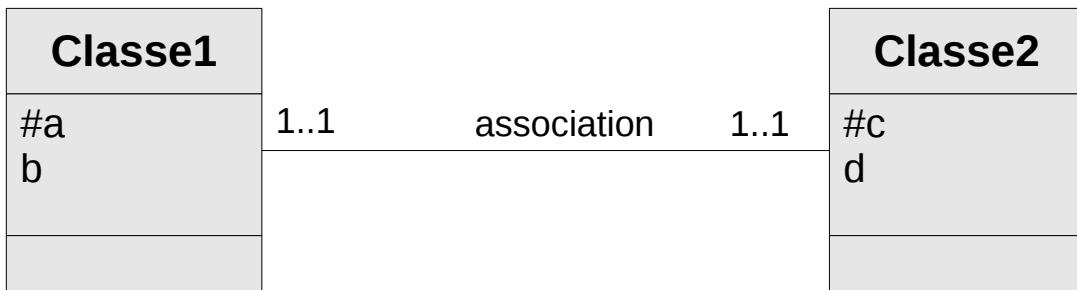
Connaître les choix possibles pour le cas de l'association 1:1 et savoir faire les bons choix.

Il existe des formulations conceptuelles en UML qui sont plus délicates à traduire au niveau logique en relationnel, comme l'association 1:1. Ces cas requièrent un choix éclairé de la part du concepteur.

3.3.1 Transformation des associations 1:1 (cas général)

Il existe deux solutions pour transformer une association 1:1 :

- Avec 2 relations : on traite l'association 1:1 une association 1:N, puis l'on ajoute une contrainte UNIQUE sur la clé étrangère pour limiter la cardinalité maximale à 1 ;
- Avec 1 relation : on fusionne les deux classes en une seule relation.



Graphique 21 Association 1:1

Méthode : Avec deux relations (clé étrangère)

- Une des deux relations est choisie pour porter la clé étrangère ;
- on ajoute les contraintes : UNIQUE ou UNIQUE NOT NULL (clé candidate) sur la clé étrangère ; et si nécessaire l'instanciation simultanée des deux relations (AND).

Classe1(#a,b,c=>Classe2) avec c UNIQUE ou clé candidate

Classe2(#c,d) avec éventuellement Classe1 AND Classe2

ou

Classe1(#a,b) avec éventuellement Classe1 AND Classe2

Classe2(#c,d,a=>Classe1) avec a UNIQUE ou clé candidate

Méthode : Avec une relation (fusion)

- On crée une seule relation contenant l'ensemble des attributs des deux classes ;
- on choisit une clé parmi les clés candidates.

Classe12(#a,b,c,d) avec c UNIQUE ou UNIQUE NOT NULL (clé candidate)

ou

Classe21(#c,d,a,b) avec a UNIQUE ou UNIQUE NOT NULL (clé candidate)

Remarque : Fusion des relations dans le cas de la traduction de l'association 1:1

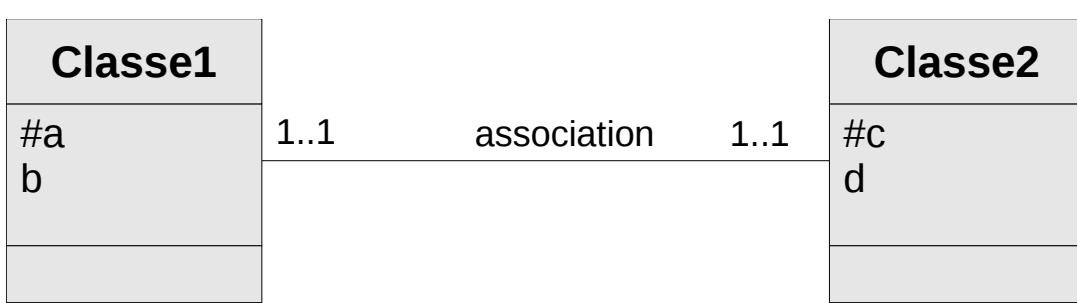
Ce choix entre les deux méthodes sera conduit par une appréciation du rapport entre :

- La complexité introduite par le fait d'avoir deux relations là où 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 sûreté factorisées ou non au niveau des deux relations,
- ...

3.3.2 Transformation des associations 1..1:1..1

Méthode : Association 1..1:1..1

- Le plus souvent c'est méthode par **fusion** des relations qui est la plus adaptée à ce cas.
- Lorsqu'elle ne l'est pas, et que l'on choisit deux relations il faut ajouter une contrainte dynamique qui contrôlera que les deux relations sont bien toujours instanciées ensemble. Notons que la clé étrangère peut être choisie comme clé primaire.



Graphique 22 Association 1:1

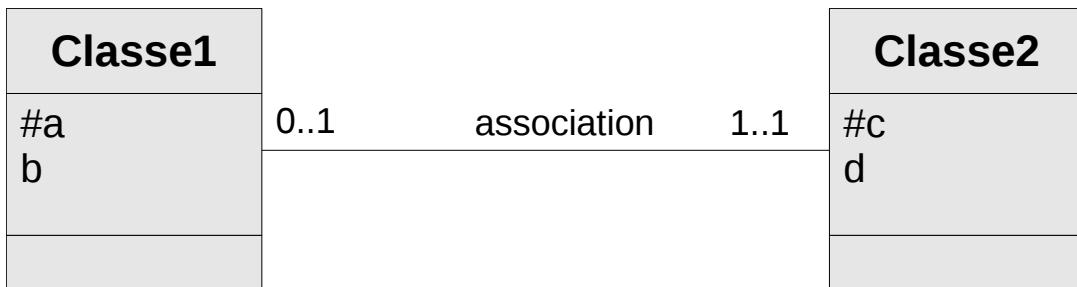
Classe12(#a,b,c,d) avec c clé candidate
ou
Classe21(#c,d,a,b) avec a clé candidate
ou
Classe1(#a,b,c=>Classe2) avec c clé candidate
Classe2(#c,d) avec Classe1 AND Classe2
ou
Classe1(#a,b) avec Classe1 AND Classe2
Classe2(#c,d,a=>Classe1) 2 avec a clé candidate

3.3.3 Transformation des associations 0..1:1..1



Méthode : Association 0..1:1..1

- Le plus souvent c'est la méthode par les deux relations qui est la plus adaptée, **on choisira toujours la relation côté 0..1 pour être référencante.**
- Il est possible d'utiliser la fusion, dans ce cas les clés côté 0..1 deviennent des attributs UNIQUE, il ne peuvent plus être clés, pouvant être NULL.



Graphique 23 Association 0..1:1

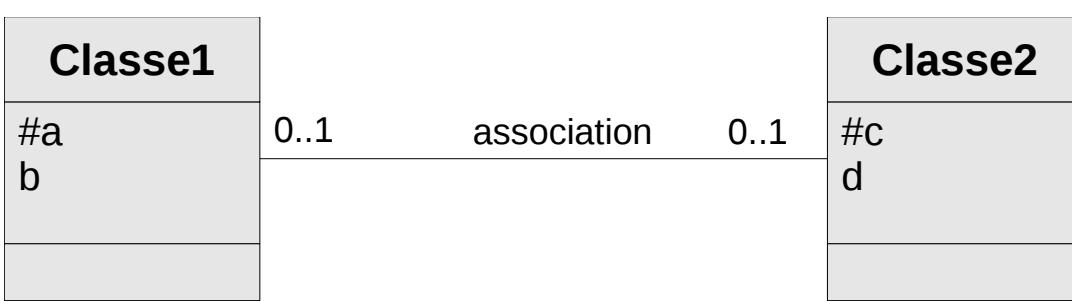
Classe1(#a,b,c=>Classe2) avec c clé candidate
Classe2(#c,d)
ou
Classe12(#a,b,c,d) avec c unique

3.3.4 Transformation des associations 0..1:0..1



Méthode : Association 0..1:0..1

- On choisit la solution avec deux relations, d'un côté ou de l'autre ; la clé étrangère est associé à la contrainte UNIQUE, ce n'est pas une clé car elle peut être nulle.
- Il n'est pas possible de choisir la fusion**, en effet l'une des deux relations pouvant être nulle, on ne pourrait plus trouver de clé.



Graphique 24 Association 0..1:0..1

Classe1(#a, b, c=>Classe2) avec c unique

Classe2(#c, d)

ou

Classe1(#a, b)

Classe2(#c, d, a=>Classe1) avec a unique

3.3.5 Exemple de choix pour une relation 1:1

Exemple

Soit deux entités, "homme" et "femme" et une association "mariage" de cardinalité 1..1:1..1 (hommes et femmes sont donc obligatoirement mariés) entre 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 correspond moins à la réalité administrative du mariage).

Bien que de type 1..1:1..1, le choix de la fusion n'est pas très opportun car il s'agit bien d'objets distincts que l'on veut modéliser. On choisira donc plutôt la représentation avec deux relations et la clé étrangère du côté de "femme". On pourra également choisir cette clé étrangère comme clé primaire si on le souhaite.

homme (#nom) avec un tuple de femme référençant obligatoirement chaque tuple de homme
femme (#nomMariage=>homme, nomJeuneFille) avec nomJeuneFille clé

Exemple

Si l'association avait été de cardinalité 0..1:0..1 (certains hommes et femmes ne sont pas mariés), un choix similaire se serait imposé, avec l'impossibilité de choisir la clé étrangère comme clé primaire, celle-ci pouvant être nulle et n'étant donc plus candidate.

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

3.4 Passage UML-Relationnel : Héritage

Objectifs

Savoir faire le passage d'un schéma conceptuel UML à un schéma relationnel.

Connaître les choix possibles pour le cas de l'héritage et savoir faire les bons choix.

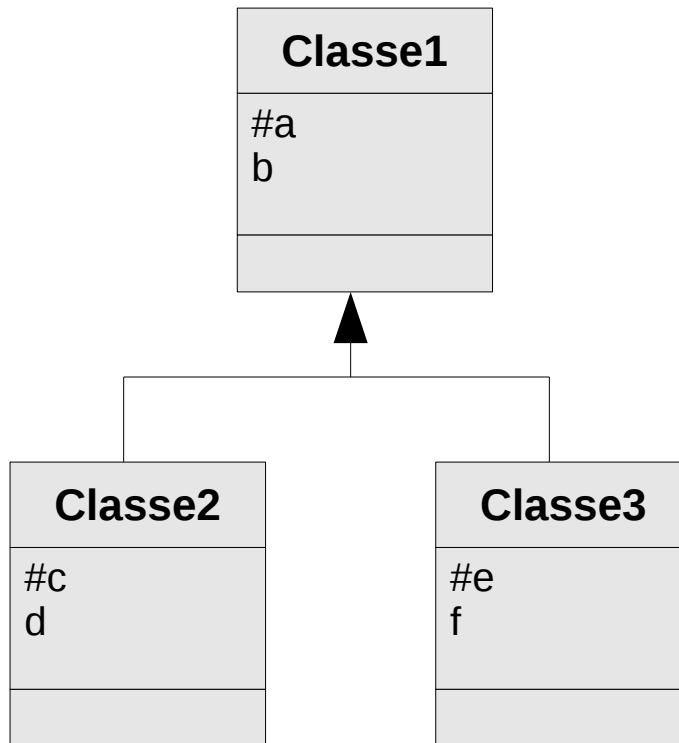
La traduction en relationnel des héritages modélisés au niveau conceptuel peut se faire de plusieurs façons, et le choix demande d'étudier plus en détail la nature de cet héritage.

3.4.1 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.



Graphique 25 Héritage



Méthode

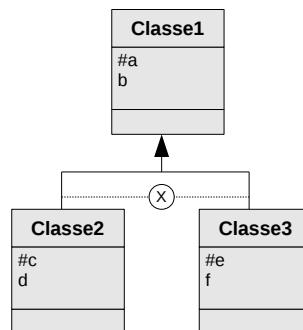
En pratique le choix entre ces trois solutions va dépendre de la réponse à trois questions :

- L'héritage est-il complet ?
- L'héritage est-il exclusif ?
- La classe mère est-elle abstraite ?



Définition : Héritage exclusif

Un héritage est exclusif si les objets d'une classe fille ne peuvent appartenir aussi à une autre classe fille. On peut le noter avec la contrainte X sur le diagramme UML.

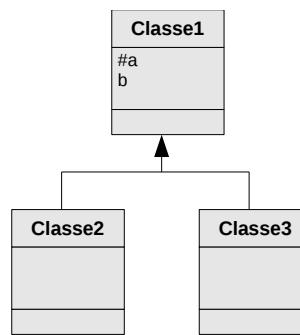


Graphique 26 Héritage exclusif



Définition : Héritage complet et presque complet

Un héritage est complet si ses classes filles n'ont aucune caractéristiques (attributs, méthodes, associations) propres.



Graphique 27 Héritage complet

Un héritage est presque complet si les classes filles ont des méthodes propres, quelques attributs propres, et **aucune association propre**.

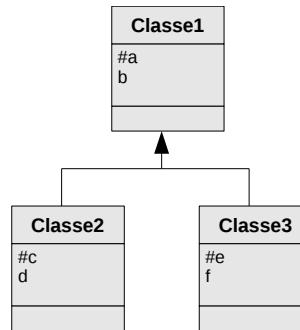
Attention

Afin de déterminer si un héritage est presque complet ou non, il faut surtout regarder les associations portant sur les classes filles, ce sont elles qui poseront le plus de problème un fois en relationnel (à cause de l'intégrité référentielle).

3.4.2 Transformation de la relation d'héritage par référence

Méthode : Héritage représenté par une référence (classe mère non abstraite)

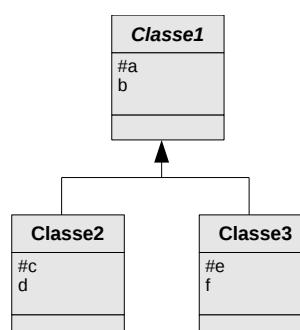
1. Chaque classe, mère ou fille, est représentée par une relation.
2. La clé primaire de la classe mère est utilisée pour identifier chacune de ses classes filles : cette clé étant pour chaque classe fille à la fois la clé primaire et une clé étrangère vers la classe mère.
3. Une vue est créée pour chaque classe fille en réalisant une jointure avec la classe mère.



Graphique 28 Héritage

Classe1(#a, b)
 Classe2(#a=>Classe1, c, d) avec c clé candidate
 vClasse2=jointure(Classe1, Classe2, a=a)
 Classe3(#a=>Classe1, e, f) avec e clé candidate
 vClasse3=jointure(Classe1, Classe3, a=a)

Méthode : Héritage représenté par une référence (classe mère abstraite)



Graphique 29 Héritage (classe mère abstraite)

- Si la classe mère est abstraite, il faut ajouter la contrainte que tous les tuples de la Classe1 sont référencés par un tuple

de la Classe2 et/ou de la Classe3.

```
Classe1(#a,b) avec Classe1 AND (Classe2 OR Classe3)
Classe2(#a=>Classe1,c,d) avec c clé candidate
vClasse2=jointure(Classe1,Classe2,a=a)
Classe3(#a=>Classe1,e,f) avec e clé candidate
vClasse3=jointure(Classe1,Classe3,a=a)
```

Remarque

- Si une classe fille a une clé primaire définie dans le modèle conceptuel, cette clé n'est pas retenue pour être la clé primaire dans le modèle relationnel, 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 instance fille référence obligatoirement une et une seule instance mère (pas d'héritage multiple) et toute instance mère est référencée une ou zéro fois (zéro fois si un objet peut être directement du type de la classe mère) par chaque instance fille .

Exemple

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 :

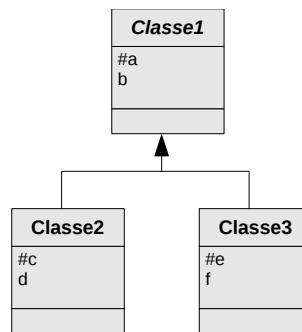
```
A (#K, A1, A2)
B (#K=>A, K', B1, B2)
VB = Jointure (A, B, A.K=B.K)
```

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 référencant dans une des classes filles.

Ainsi dans l'exemple précédent, un A peut être instancié en toute indépendance par rapport à la relation B.

3.4.3 Transformation de la relation d'héritage par les classes filles

 Méthode : Héritage absorbé par les classes filles (classe mère abstraite)



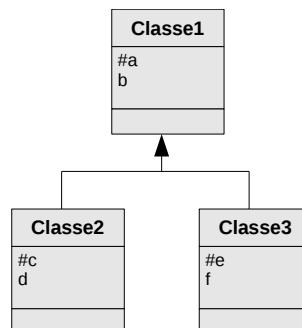
Graphique 30 Héritage (classe mère abstraite)

- Chaque classe fille est représentée par une relation, la classe mère n'est pas représentée (elle est abstraite).
- Tous les attributs de la classe mère sont répétés au niveau de chaque classe fille.
- La clé primaire de la classe mère est utilisée pour identifier chacune de ses classes filles.

Classe2(#a,b,c,d) avec c clé candidate

Classe3(#a,b,e,f) avec e clé candidate

 Méthode : Héritage absorbé par les classes filles (classe mère non abstraite)



Graphique 31 Héritage

- On crée une relation supplémentaire pour gérer les objets de la classe mère
- On ajoute une contrainte qui exprime que les tuples de la classe mère ne peuvent exister dans les classes filles
- On ajoute une vue pour représenter tous les objets de la classe mère (par union des tuples de la classe mère et des classes filles).

```
Classe1(#a,b) avec Classe1 X Classe2 et Classe1 X Classe3
vClasse1=union(union(Classe1,projection(Classe2,a,b)),projection(Classe3,a,b))
Classe2(#a,b,c,d) avec c clé candidate
Classe3(#a,b,e,f) avec e clé candidate
```

Remarque

- Si une classe fille a une clé primaire au niveau du MCD, 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).

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)
vA = Union (Projection (B, K, A1, A2), Projection (C, K, A1, A2))
```

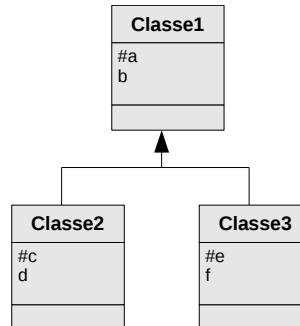
Si A n'avait pas été abstraite elle aurait donné naissance à une relation A possédant les attributs A1 et A2 et qui n'aurait alors contenu que les tuples qui ne sont ni des B ni des C.

Attention: 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.

3.4.4 Transformation de la relation d'héritage par la classe mère

 Méthode : Héritage absorbé par la classe mère (classe mère non abstraite)



Graphique 32 Héritage

1. Seule la classe mère est représentée par une relation (ses classes filles ne sont pas représentées par des relations).
2. Tous les attributs de chaque classe fille sont réintégrés au niveau de la classe mère.
3. La clé primaire de la classe mère est utilisée pour identifier la relation.
4. Un attribut supplémentaire, dit de discrimination, est ajouté à la classe mère, afin de distinguer les tuples :
 - cet attribut est de type énumération et a pour valeurs possibles les noms de la classe mère ou des différentes classes filles ;
 - afin de gérer l'héritage non exclusif (un objet peut être de plusieurs classes filles à la fois), le domaine de l'attribut de discrimination, peut être étendu à des combinaisons de noms des différentes classes filles.
5. Chaque classe est représentée par une vue qui restreint aux tuples de la relation correspondants et les projette sur les attributs correspondants.

```
Classe1(#a,b,c,d,e,f,t:{1,2,3,23}) avec c et e uniques
```

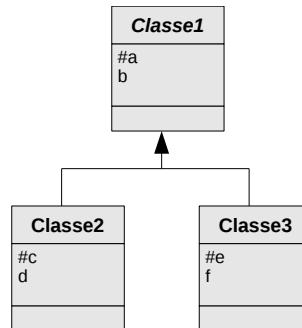
```
vClasse1=projection(restriction(Classe1,t=1),a,b)
```

```
vClasse2=projection(restriction(Classe1,t=2 ou t=23),a,b,c,d)
```

```
vClasse3=projection(restriction(Classe1,t=3 ou t=23),a,b,e,f)
```



Méthode : Héritage absorbé par la classe mère (classe mère abstraite)



Graphique 33 Héritage (classe mère abstraite)

- Si la classe mère est abstraite, sa valeur est ôtée de l'attribut de discrimination
- Une contrainte supplémentaire doit vérifier que soit c soit e est obligatoirement valué (ou les deux).

Classe1(#a,b,c,d,e,f,t:{2,3,23}) avec c et e uniques et c ou e valués

vClasse2=projection(restriction(Classe1,t=2 ou t=23),a,b,c,d)

vClasse3=projection(restriction(Classe1,t=3 ou t=23),a,b,e,f)



Remarque

- 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 elle n'officiera pas en tant que clé candidate car elle pourra contenir des valeurs nulles (elle sera néanmoins unique).



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 :

```

A (#K, A1, A2, B1, B2, C1, C2, D:{'B', 'C', 'BC'})
vB = Projection (Restriction (A, D='B' ou 'BC'), K, A1, A2, B1, B2)
vC = Projection (Restriction (A, D='C' ou 'BC'), K, A1, A2, C1, C2)
  
```

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 il est moins naturel de disposer d'une table associée à cette classe.



Conseil : 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 selon cette approche, a fortiori si la classe mère n'est pas abstraite.



Conseil : 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".

3.4.5 Éléments de choix



Méthode : 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
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 Employé 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).
Par les classes filles	Redondance liée à l'existence 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).
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)

Tableau 10 Avantages et inconvénients de chaque transformation

Mode de transformation	Classe mère abstraite	Classe mère non abstraite	Héritage exclusif	Héritage non exclusif	Héritage complet	Héritage presque complet	Héritage non complet
Par référence	Contrôle	OK	Contrôle	OK	Inadapté	Mal adapté	OK
Par les classes filles	OK	Compliqué	OK	Problème	Inadapté	Mal adapté	OK
Par la classe mère	Mal adapté	OK	Contrôle	Compliqué	OK	OK	Problème

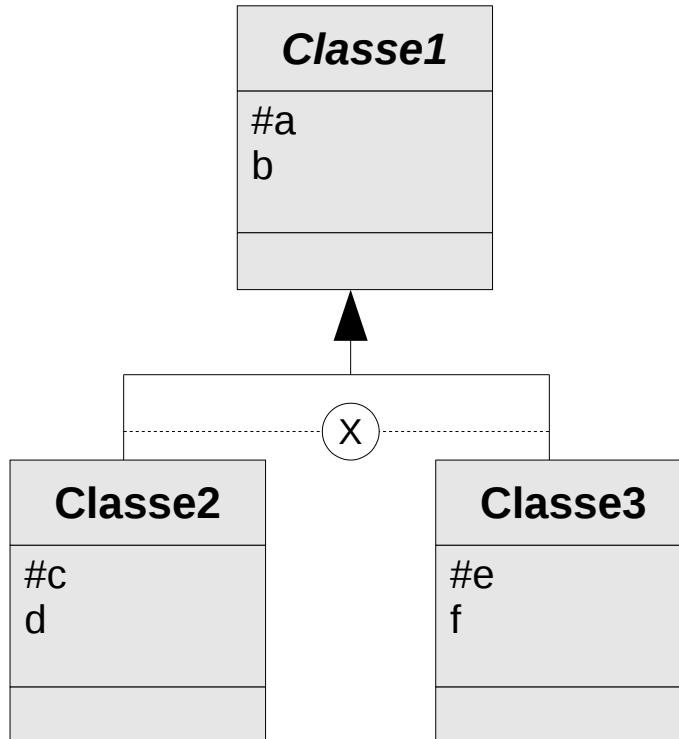
Tableau 11 Choix de la bonne transformation

3.4.6 Les cas simples



Méthode : Cas de l'héritage exclusif avec classe mère abstraite

Dans ce cas, toujours choisir un **héritage par les classes filles**.



Graphique 34 Héritage exclusif (classe mère abstraite)

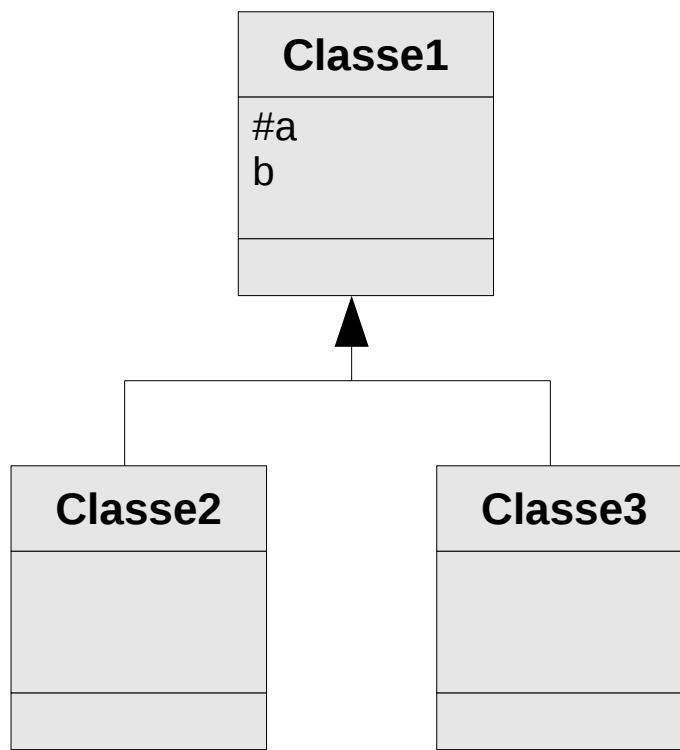
Classe2(#a, b, c, d) avec c clé candidate

Classe3(#a, b, e, f) avec e clé candidate



Méthode : Cas de l'héritage complet

Si l'héritage n'est pas exclusif ou que la classe mère n'est pas abstraite, toujours choisir un **héritage par la classe mère**.



Graphique 35 Héritage complet

1. Si la classe mère est abstraite :

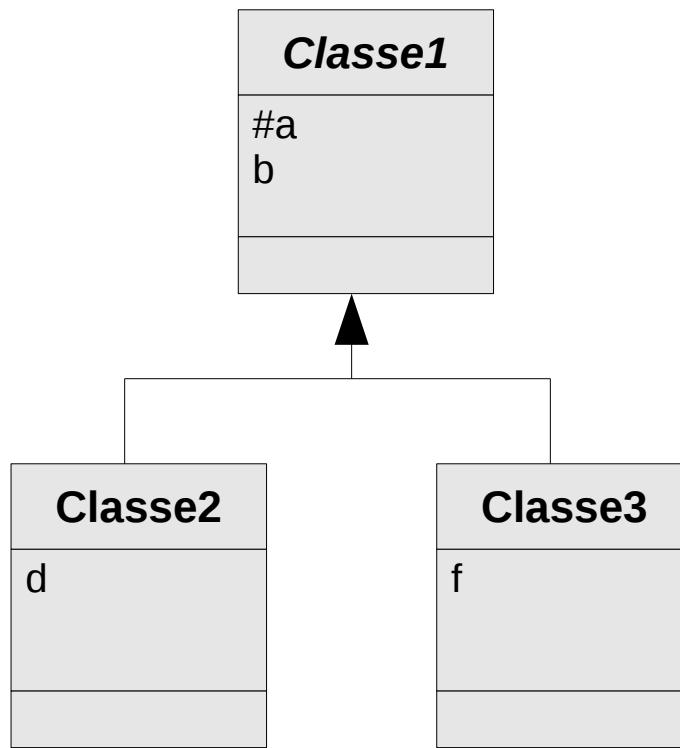

```
Classe1(#a, b, t:{2,3,23})
vClasse2=restriction(Classe1, t=2 ou t=23))
vClasse3=restriction(Classe1, t=3 ou t=23))
```
2. Si la classe mère n'est pas abstraite :


```
Classe1(#a, b, t:{1,2,3,23})
vClasse1=restriction(Classe1, t=1)
vClasse2=restriction(Classe1, t=2 ou t=23))
vClasse3=restriction(Classe1, t=3 ou t=23))
```



Méthode : Cas de l'héritage presque complet

L'héritage presque complet peut être géré comme l'héritage complet, **par la classe mère**, surtout si les classes filles ne possèdent pas de clé propre.



Graphique 36 Héritage presque complet (classe mère abstraite)

1. Si la classe mère est abstraite :


```

Classe1(#a, b, d, f, t:{2, 3, 23})
vClasse2=projection(restriction(Classe1, t=2 ou t=23), a, b, d)
vClasse3=projection(restriction(Classe1, t=3 ou t=23), a, b, f)
      
```
2. Si la classe mère n'est pas abstraite :

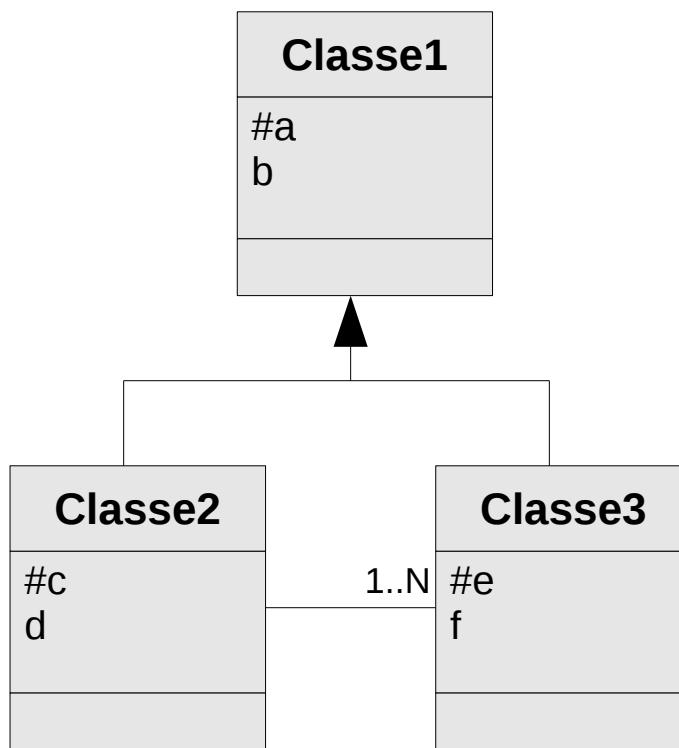

```

Classe1(#a, b, c, d, e, f, t:{1, 2, 3, 23})
vClasse1=projection(restriction(Classe1, t=1), a, b)
vClasse2=projection(restriction(Classe1, t=2 ou t=23), a, b, d)
vClasse3=projection(restriction(Classe1, t=3 ou t=23), a, b, f)
      
```



Méthode : Cas de l'héritage non complet et non exclusif avec classe mère non abstraite

Dans ce cas, choisir un **héritage par référence**.



Graphique 37 Héritage non exclusif et non complet

```

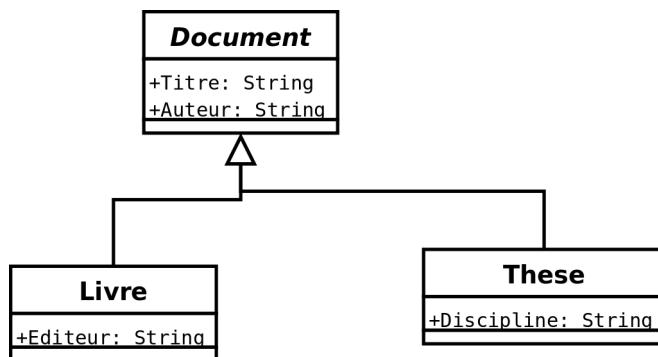
Classe1(#a, b)
Classe2(#a=>Classe1, c, d) avec c clé candidate
vClasse2=jointure(Classe1, Classe2, a=a)
Classe3(#a=>Classe1, e, f, c=>Classe2) avec e clé candidate
vClasse3=jointure(Classe1, Classe3, a=a)
  
```

3.4.7 Exemple de transformation d'une relation d'héritage



Exemple

Soit le modèle UML suivant :



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 (c'est le cas si une thèse peut-être publiée comme un livre par exemple), 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). Mais l'on peut considérer qu'il **est presque complet**, car il n'y a qu'un attribut par classe fille de différence et pas d'association associée aux classes filles.

On pourrait chercher à appliquer le cas général (ni exclusif, ni complet) et appliquer une traduction de l'héritage par référence. Néanmoins, comme **la classe mère est abstraite**, l'héritage par référence demandera un contrôle assez compliqué.

La meilleure solution est donc ici un **héritage par la classe mère**. Observons néanmoins les avantages et inconvénients que chacun des trois choix porterait.

3.4.7.1 Héritage représenté par une référence

```
Document(#Titre:Chaîne, Auteur:Chaîne) avec Document AND (These OR Livre)
These(#Titre=>Document, Discipline:Chaîne)
Livre(#Titre=>Document, Editeur:Chaîne)
vThese = Jointure (Document, These, Document.Titre=These.Titre)
vLivre = Jointure (Document, Livre, Document.Titre=Livre.Titre)
```

 Remarque : Avantages du choix

Il n'y a ni redondance ni valeur nulle inutile dans les relations These et Livre.

 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.

De plus la classe Document étant abstraite, il ne devra jamais y avoir de tuple dans document qui n'a pas de tuple correspondant dans Livre ou These. Ceci n'est pas contrôlable directement en relationnel et devra donc être vérifié au niveau applicatif (ou bien la classe ne devra plus être considérée comme abstraite).

3.4.7.2 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.

3.4.7.3 Héritage absorbé par la classe mère

```
Document(#Titre, Discipline:Chaîne, Editeur:Chaîne, Auteur:Chaîne, Type:{These|Livre|These+Livre})
vThese = Projection (Restriction (Document, Type='These' ou 'These+Livre'), Titre,
Discipline, Auteur)
vLivre = Projection (Restriction (Document, Type='Livre' ou 'These+Livre'), Titre,
Editeur, Auteur)
```

 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, et il n'introduit pas de redondance.

 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.

3.4.8 Transformations de l'héritage : Synthèse

L'héritage est toujours délicat à traduire en relationnel, ce qui est dommage car son pouvoir de représentation conceptuel est fort.



Méthode : Algorithme de choix

Un conseil pour assurer 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 l'algorithme suivant :

- SI classe mère abstraite ALORS
 - SI héritage exclusif ALORS par les classes filles
 - SINON
 - SI héritage complet ou presque complet ALORS par la classe mère
 - SINON problème (aucune bonne solution, chercher la moins mauvaise)
- SINON
 - SI héritage complet ou presque complet ALORS héritage par la classe mère
 - SINON
 - SI héritage non exclusif ALORS par référence
 - SINON problème

Abs	X	C	
OUI	OUI	OUI	Fille
OUI	OUI	NON	Fille
OUI	NON	OUI	Mère
OUI	NON	NON	?
NON	OUI	OUI	Mère
NON	OUI	NON	?
NON	NON	OUI	Mère
NON	NON	NON	Référence

Tableau 12 Table de décision

Conseil



On retiendra que :

- Les héritages complets ne posent jamais problème
- Les deux cas les plus délicats sont, quand l'héritage n'est pas complet :
 - a. héritage exclusif et classe mère non abstraite
 - b. héritage non exclusif et classe mère abstraite



Attention: Penser aux vues !

Il est important de bien penser à ajouter les vues permettant de retrouver le schéma initialement recherché.



Méthode : 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.

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.

3.5 Le passage E-A vers Relationnel

Le passage E-A vers relationnel est très similaire au passage UML vers relationnel. Les règles décrites ici sont donc à compléter avec celles prescrites pour UML.

3.5.1 Transformation des entités



Méthode : 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.



Méthode : 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.



Rappel : Voir aussi

Transformation des classes (cf. Transformation des classes p 43)

Transformation des compositions (cf. Transformation des compositions p 49)

3.5.2 Transformation des associations



Méthode : 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.



Méthode : 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.



Méthode : Association 1:1

Une association 1:1 est gérée comme un cas particulier d'association 1:N, en ajoutant une contrainte d'unicité à la clé étrangère migrée.



Remarque : Cardinalité minimale 0 ou 1

- Selon que la cardinalité minimale est 0 ou 1 (ou plus) du côté de la relation référencante 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'existence de tuples référençant pour chaque tuple de la relation référencée.



Rappel : Voir aussi

Transformation des associations 1:N (cf. Transformation des associations 1:N p 45)

Transformation des associations N:M (cf. Transformation des associations N:M p 46)

Transformation des associations 1:1 (cas général) (cf. Transformation des associations 1:1 (cas général) p 52)

3.5.3 Transformation des attributs



Méthode : 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.



Méthode : 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.



Méthode : 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.



Méthode : 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.



Rappel : Voir aussi

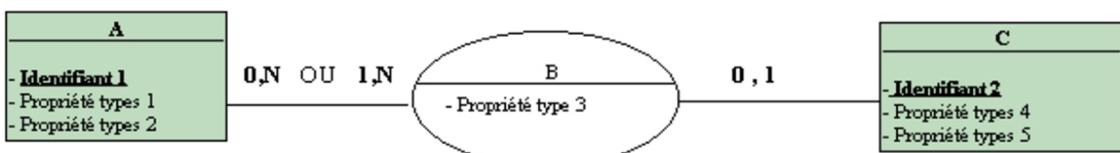
Transformation des attributs (cf. Transformation des attributs p 44)

Transformation des attributs dérivés et méthodes (cf. Transformation des attributs dérivés et méthodes p 45)

3.5.4 Exemple de passage E-A vers relationnel



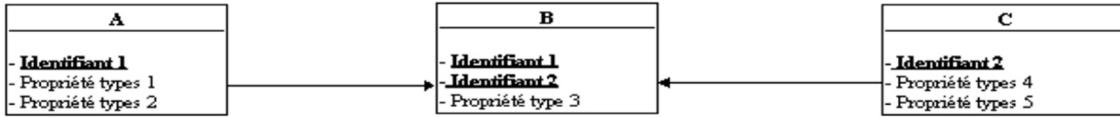
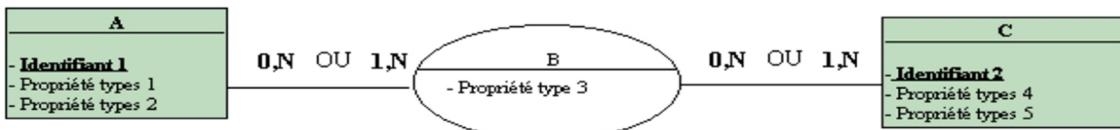
Exemple



Transformation avec une relation 1:N



Exemple



Transformation avec une relation M:N

3.5.5 Transformation de la relation d'héritage

Trois solutions existent pour transformer une relation d'héritage exprimée en E-A :

- par référence,
- par absorption par les sous-types d'entité,
- par absorption par l'entité générale.



Attention: Les entités non finales sont abstraites

En modélisation E-A on considérera 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.



Rappel : Voir aussi

Transformation de la relation d'héritage par référence (cf. Transformation de la relation d'héritage par référence p 56)

Transformation de la relation d'héritage par les classes filles (cf. Transformation de la relation d'héritage par les classes filles p 57)

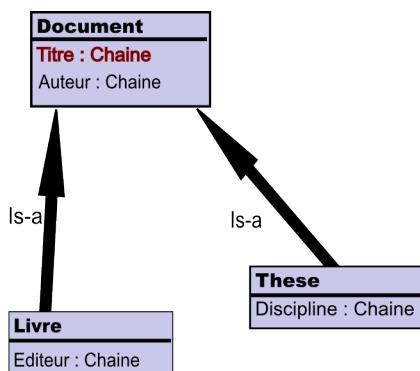
Transformation de la relation d'héritage par la classe mère (cf. Transformation de la relation d'héritage par la classe mère p 58)

3.5.6 Exemple de transformation d'une relation d'héritage



Exemple

Soit le modèle E-A suivant :



Représentation de documents

Héritage représenté par une référence

```

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

Héritage absorbé par les sous-types d'entité

```

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

Héritage absorbé par l'entité générale

```

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

3.5.7 Comparaison des modèles E-A, UML et relationnel

Modèle E-A	Modèle UML	Modèle relationnel
Entité	Classe	Relation
Occurrence 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
Occurrence d'une association	Occurrence d'une association	Nuplet
Héritage	Héritage	Vue

Tableau 13 Passage E-A et UML vers Relationnel

3.6 Algèbre relationnelle

Objectifs

Connaître les opérateurs relationnels.
Maîtriser l'algèbre relationnelle.

3.6.1 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 additionnels**
 - 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.

3.6.2 Opérateurs ensemblistes

 Attention

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

 Définition : 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.

 Définition : 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.

 Définition : 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 équivalente 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.

3.6.3 Projection

 Définition : 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 : Élimination des doublons

Après suppression d'une partie des attributs du schéma, la relation peut comporter des doublons. Étant 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)

3.6.4 Restriction

 Définition : 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 :

$R = \text{Restriction} (\text{Personne}, \text{Age} > 25)$

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

(Durand, Jean, 30)

3.6.5 Produit



Définition : 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.

Synonymes : 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 :

$R = \text{Produit} (\text{Homme}, \text{Femme})$

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)

3.6.6 Jointure



Définition : 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 :

$R = \text{Jointure} (\text{Homme}, \text{Enfant}, \text{Homme.Nom}=\text{Enfant.Nom})$

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.

3.6.7 Jointure naturelle

 Définition : 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 $\text{Jointure}(R1, R2, R1.A=R2.A)$ est équivalente à l'opération $\text{JointureNaturelle}(R1, R2)$.

 Remarque

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

3.6.8 Jointure externe

 Introduction

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.

 Définition : 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.

 Définition : 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.

Synonymes : Jointure gauche

 Définition : 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.

Synonymes : 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 :

$R = \text{JointureExterne} (\text{Homme}, \text{Enfant}, \text{Homme.Nom}=\text{Enfant.Nom})$

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.

3.6.9 Division



Définition : 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 :

$R = \text{Division} (\text{Homme}, \text{Métier})$

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.

3.6.10 Proposition de notations

Introduction

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

$R = \text{Union} (R1, R2)$
 $R = \text{Différence} (R1, R2)$

```
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)
```

3.6.11 Exercice de synthèse : Opérateurs de base et additionnels

Réécrivez les opérateurs additionnels suivants, à partir d'opérateurs de base :

Question 1

Réécrivez Intersection à partir de Différence

Question 2

Réécrivez Jointure à partir de Produit et Restriction

Question 3

Réécrivez Division à partir de Produit, Restriction et Difference

3.7 En résumé : Schéma relationnel

Schéma relationnel

Un schéma relationnel permet une formalisation d'un modèle logique.

- Relation ou table
 - Sous-ensemble d'un produit cartésien
 - Attribut ou colonne
 - Prend ses valeurs dans un domaine
 - Enregistrement ou ligne
 - Pose une valeur (y compris la valeur "null") pour chaque attribut
 - Clé
 - Groupe d'attributs ayant un rôle d'identification au sein d'un enregistrement
 - Clé candidate
 - Identifie de façon unique un enregistrement
 - Clé primaire
 - Clé candidate choisie pour représenter un enregistrement pour sa facilité d'usage
 - Clé étrangère
 - Référence la clé primaire d'un tuple d'une autre relation pour exprimer un lien

3.8 Bibliographie commentée sur le modèle relationnel

 Complément: Synthèses

SQL2 SQL3, applications à Oracle [Delmal01]

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

SQL est un langage standardisé, implémenté par tous les SGBDR, qui permet, indépendamment de la plate-forme technologique et de façon déclarative, de définir le modèle de données, de le contrôler et enfin de le manipuler.

4.1 Qu'appelle-t-on SQL?



Définition : SQL

SQL (pour langage de requêtes structuré) est un langage déclaratif destiné à la manipulation de bases de données au sein des SGBD et plus particulièrement des SGBDR.

SQL est un langage déclaratif, il n'est donc pas 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 Langage de Définition de Données (LDD, ou en anglais DDL Data Definition Language) pour créer et supprimer des objets dans la base de données (tables, contraintes d'intégrité, vues, etc.).
- Le Langage de Contrôle de Données (LCD, ou en anglais DCL, Data Control Language) pour gérer les droits sur les objets de la base (création des utilisateurs et affectation de leurs droits).
- Le Langage de Manipulation de Données (LMD, ou en anglais DML, Data Manipulation Language) 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.



Complément : Origine du SQL

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 aux États-Unis par l'ANSI pour donner SQL/86 (puis au niveau international par l'ISO en 1987).



Complément : Versions de SQL

- SQL-86 (ou SQL-87) : Version d'origine
- SQL-89 (ou SQL-1) : Améliorations mineures
- SQL-92 (ou SQL-2) : Extensions fonctionnelles majeures (types de données, opérations relationnelles, instruction LDD, transactions, etc.)
- SQL-99 (ou SQL-3) : Introduction du PSM (couche procédurale sous forme de procédure stockées) et du ROD
- SQL-2003 : Extensions XML
- SQL-2006 : Améliorations mineures (pour XML notamment)
- SQL-2008 : Améliorations mineures (pour le ROD notamment)



Remarque : Version SQL et implémentations SGBD

Selon leur niveau d'implémentation de SQL, les SGBD acceptent ou non certaines fonctions.

Certains SGBD ayant entamé certaines implémentations avant leur standardisation définitive, ces implémentations peuvent différer de la norme.

4.2 Le Langage de Définition de Données de SQL

Objectifs

Maîtriser les bases du SQL pour créer et modifier des tables et des vues.

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.

4.2.1 Types de données

Introduction

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.

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.

Les types chaîne de caractères

On distingue principalement les types CHAR(X) et VARCHAR(X), où X est obligatoire et désigne la longueur de la chaîne. CHAR définit des chaînes de longueur fixe (complétée à droite par des espaces, si la longueur 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.

Les types date

Les types date sont introduits avec la norme SQL2. On distingue le type DATE qui représente une date selon un format de type "AAAA-MM-JJ" et le type 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.

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 Object) pour représenter des données binaires tels que des documents multimédia (images bitmap, vidéo, etc.).

La valeur NULL

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.

Par défaut en SQL NULL fait partie du domaine, il faut l'exclure explicitement par la clause NOT NULL après la définition de type, si on ne le souhaite pas.

4.2.2 Crédation de tables

Introduction

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

 Définition : Crédation 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> [NOT NULL],
    <nom colonne2> <type colonne2> [NOT NULL],
    ...
    <nom colonneN> <type colonneN> [NOT NULL]
);
```



Exemple

```
CREATE TABLE Personne (
    Nom VARCHAR(25) NOT NULL,
    Prenom VARCHAR(25),
    Age INTEGER(3)
);
```



Attention: Contrainte d'intégrité

La définition des types 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.

4.2.3 Contraintes d'intégrité



Définition : 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 BDI.

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
- 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'attributs>) : définit les attributs de la liste comme la clé primaire
- UNIQUE (<liste d'attributs>) : interdit que deux tuples de la relation aient les mêmes valeurs pour l'ensemble des attributs de la liste.
- FOREIGN KEY (<liste d'attributs>) 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> [NOT NULL] <contraintes colonne1>,
    <nom colonne2> <type colonne2> [NOT NULL] <contraintes colonne2>,
    ...
    <nom colonneN> <type colonneN> [NOT NULL] <contraintes colonneN>,
    <contraintes de 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),
    UNIQUE (Nom, Prenom)
```

);

 Remarque : Clé candidate

La clause 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.

4.2.4 Exemple de contraintes d'intégrité

 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.

 Exemple : Réécriture avec uniquement des contraintes de table

```
CREATE TABLE Personne (
    N°SS CHAR(13),
    Nom VARCHAR(25) NOT NULL,
    Prenom VARCHAR(25) NOT NULL,
    Age INTEGER(3),
    Mariage CHAR(13),
    Codepostal INTEGER(5),
    Pays VARCHAR(50),
    PRIMARY KEY (N°SS),
    UNIQUE (Nom, Prenom),
    CHECK (Age BETWEEN 18 AND 65),
    FOREIGN KEY (Mariage) REFERENCES Personne(N°SS),
    FOREIGN KEY (Codepostal, Pays) REFERENCES Adresse (CP, Pays)
);
CREATE TABLE Adresse (
    CP INTEGER(5) NOT NULL,
    Pays VARCHAR(50) NOT NULL,
    Initiale CHAR(1),
    PRIMARY KEY (CP, Pays),
    CHECK (Initiale = LEFT(Pays, 1))
);
```

Ce schéma est strictement le même que le précédent, simplement les contraintes ont toutes été réécrites comme des contraintes de table.

4.2.5 Création de vues



Définition : 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.



Rappel : Vues et héritage

Les vues sont particulièrement utiles pour restituer les relations d'héritage perdues lors de la transformation MCD vers MLD.

4.2.6 Suppression 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;
```

4.2.7 Modification de tables



Introduction

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 d'une 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 supprimer et recréer les tables référencantes (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

4.2.8 Exemple de modifications de tables

Table initiale

Soit une table initiale 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)
);
```

4.3 Gestion avec le Langage de Manipulation de Données de SQL

Objectifs

Maîtriser les bases du SQL pour entrer, modifier et effacer des données dans les tables.

4.3.1 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 informations 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 spécifiées 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 recréédités pour le même montant (et à la même date), avec la mention annulation dans l'objet du crédit. Ceci pourrait typiquement être réalisé en cas de débits erronés 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).

4.3.2 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'
```

4.3.3 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'
```

4.4 Questions avec le Langage de Manipulation de Données de SQL

Objectifs

Maîtriser les bases du SQL pour écrire des questions exigeant des jointures, projections, restriction, des tris et des agrégats.

4.4.1 Sélection

Introduction

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



Définition : 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 prédictat et par conséquent renvoie un booléen. Cette partie est optionnelle.

Afin de décrire un attribut d'une relation en particulier 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 sans 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'.

4.4.2 Opérateurs de comparaisons et opérateurs logiques

Introduction

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.



Définition : 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

 Remarque : Opérateur LIKE

L'opérateur LIKE 'chaîne' permet d'insérer des jokers dans l'opération de comparaison (alors que l'opérateur = teste une égalité stricte) :

- Le joker % désigne 0 ou plusieurs caractères quelconques
- Le joker _ désigne 1 et 1 seul caractère

On préférera l'opérateur = à l'opérateur LIKE lorsque la comparaison n'utilise pas de joker.

4.4.3 Expression du produit cartésien

 Syntaxe

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

 Exemple

R1	#A	B	C=>R2
1	Alpha	10	
2	Bravo	10	
3	Charlie	20	
4	Delta		

R2	#X	Y
10	Echo	
20	Fox	
30	Golf	

A	B	C	X	Y
1	Alpha	10	10	Echo
1	Alpha	10	20	Fox
1	Alpha	10	30	Golf
2	Bravo	10	10	Echo
2	Bravo	10	20	Fox
2	Bravo	10	30	Golf
3	Charlie	20	10	Echo
3	Charlie	20	20	Fox
3	Charlie	20	30	Golf
4	Delta		10	Echo
4	Delta		20	Fox
4	Delta		30	Golf

Produit(R1, R2)

```
SELECT *
```

Tableau 14 Exemple de produit (SQL et Algèbre)

4.4.4 Expression d'une projection

 Syntaxe

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



Exemple

R1	#A	B	C=>R2
1	Alpha	10	
2	Bravo	10	
3	Charlie	20	
4	Delta		

R	A	C
1	10	
2	10	
3	20	

Projection(R1, A, C)

SELECT A, C

Tableau 15 Exemple de projection (SQL et Algèbre)

4.4.5 Expression d'une restriction



Syntaxe

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



Exemple

R1	#A	B	C=>R2
1	Alpha	10	
2	Bravo	10	
3	Charlie	20	
4	Delta		

R	A	B	C
1	Alpha	10	
2	Bravo	10	

Restriction(R1, C<20)

SELECT *
FROM R1

Tableau 16 Exemple de restriction (SQL et Algèbre)

4.4.6 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 :

```
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

R1	#A	B	C=>R2	R2	#X	Y
	1	Alpha	• 10		► 10	Echo
	2	Bravo	• 10		► 20	Fox
	3	Charlie	20		► 30	Golf
	4	Delta				

R	A	B	C	X	Y
	1	Alpha	10	10	Echo
	2	Bravo	10	10	Echo
	3	Charlie	20	20	Fox

Jointure(R1, R2, R1.C=R2.X)

```
SELECT *
FROM R1 INNER JOIN R2
```

Tableau 17 Exemple de jointure (SQL et Algèbre)



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 renommage 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
```

4.4.7 Expression d'une jointure externe



Syntaxe : Jointure externe, gauche ou droite

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



Exemple : Jointure externe gauche

```
SELECT Num
FROM Avion LEFT OUTER JOIN Vol
```

ON Avion.Num=Vol.Num

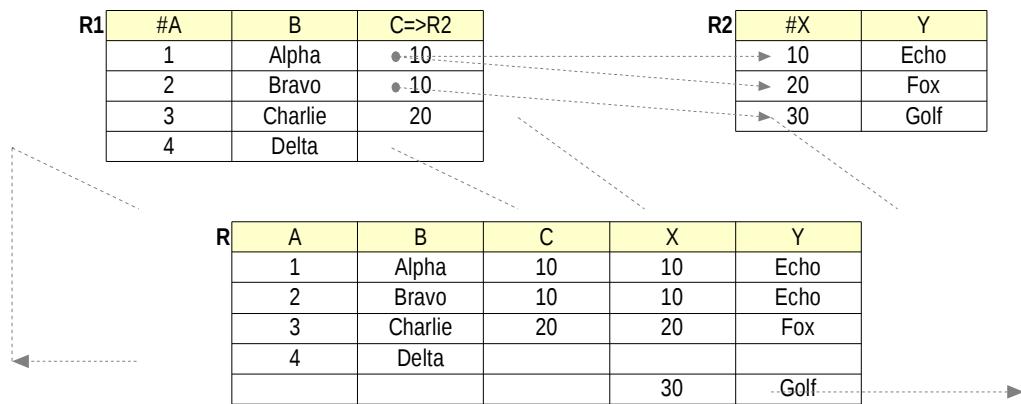
Cette requête permet de sélectionner tous les avions, y compris ceux non affectés à un vol.

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 tuples 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".

Exemple

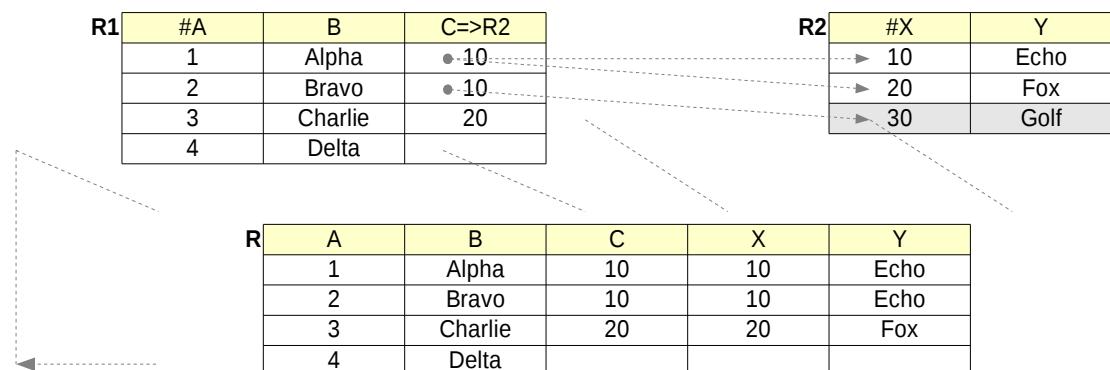


JointureExterne(R1, R2, R1.C=R2.X)

```
SELECT *
FROM R1 OUTER JOIN R2
```

Tableau 18 Exemple de jointure externe (SQL et Algèbre)

Exemple



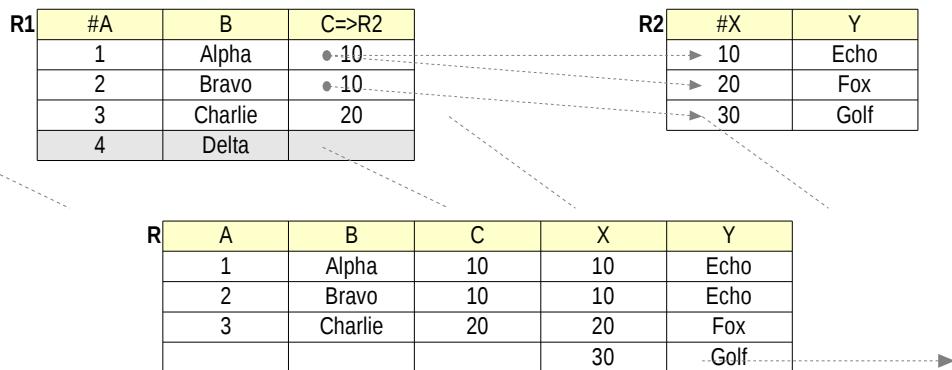
JointureExterneGauche(R1, R2, R1.C=R2.X)

```
SELECT *
FROM R1 LEFT OUTER JOIN R2
```

Tableau 19 Exemple de jointure externe gauche (SQL et Algèbre)



Exemple



JointureExterneDroite(R1,R2,R1.C=R2.X)

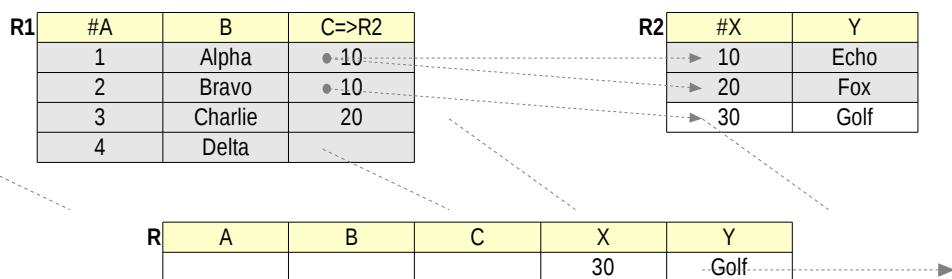
```
SELECT *
FROM R1 RIGHT OUTER JOIN R2
```

Tableau 20 Exemple de jointure externe droite (SQL et Algèbre)



Méthode : Trouver les enregistrements non joints

La jointure externe sert en particulier pour trouver les enregistrements d'une table qui ne sont pas référencés par une clé étrangère. Il suffit de sélectionner, après la jointure externe, tous les enregistrements pour lesquels la clé de la relation référencante est nulle, on obtient alors ceux de la relation référencée qui ne sont pas référencés.



```
Restriction(
    JointureExterneDroite(R1,R2,R1.C=R2.X),
    R1.A IS NULL)
```

```
SELECT *
FROM R1 RIGHT OUTER JOIN R2
ON R1.C=R2.X
```

Tableau 21 Exemple de sélection d'enregistrements non référencés (SQL et Algèbre)

4.4.8 Opérateurs ensemblistes

Introduction

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. Certains SGBD sont susceptibles de ne pas les implémenter.

4.4.9 Tri

Introduction

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
```

4.4.10 Fonctions de calcul



Définition : 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 clause 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.

4.4.11 Agrégats



Définition : 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.



Syntaxe

```
SELECT <liste d'attributs de partitionnement 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 calcule 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.



Attention: 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 partitionnement).



Remarque : Fonctions de calcul sans partitionnement

Si une ou plusieurs fonctions de calcul sont appliquées sans partitionnement, 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).

4.5 Instructions avancées pour le LMD de SQL

Objectifs

Être capable d'apprendre des notions particulières de SQL liés à un SGBD en particulier ou à des évolutions futures de SQL.

4.5.1 Requêtes imbriquées

Introduction

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.



Définition : 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)
```

4.5.2 Sous-requête d'existence IN

Introduction

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 préférera 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.

4.5.3 Sous-requête d'existence EXISTS

Introduction

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.

4.5.4 Sous-requête de comparaison ALL

Introduction

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)
```

4.5.5 Sous-requête de comparaison ANY

Introduction

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.

4.5.6 Raffinement de questions dans la clause FROM

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)
```

4.6 Le Langage de Contrôle de Données de SQL

Objectifs

Maîtriser les bases du SQL pour attribuer et révoquer des droits sur des objets d'une base de données.

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.

4.6.1 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 : Crédit des utilisateurs

Les modalités de création d'utilisateurs, voire de groupes d'utilisateurs dans le SGBD, reste dépendantes de celui-ci. Des commandes 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.

4.6.2 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 FROM Pierre;  
REVOKE ALL PRIVILEGES ON Adresse FROM 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 avaient obtenu ce même droit par l'utilisateur en question.

4.6.3 Création d'utilisateurs

Le standard SQL laisse la gestion des utilisateurs à la discréction du SGBD.

Néanmoins la commande CREATE USER est couramment proposée (Oracle, Postgres, ...).

4.7 En résumé : SQL

Langage SQL

Le langage SQL permet la création, le contrôle et la manipulation d'une BD.

- LDD
 - Permet de créer, modifier et supprimer les objets d'une BD
 - CREATE TABLE
 - CREATE VIEW
- LCD
 - Permet de définir les droits des utilisateurs sur les objets de la BD
 - GRANT
 - REVOKE
- LMD
 - Permet d'entrer et sortir des données dans la BD
 - INSERT, UPDATE, DELETE
 - SELECT

4.8 Bibliographie commentée sur le SQL



Complément: Exercice à faire

Tutoriel SQL [w-univ-lyon2.fr/~jdarmont]

Un excellent exerciceur permettant de poser des questions en SQL à une base de données en temps réel. 18 questions à faire absolument.



Complément: Pour continuer de s'exercer

Initiation à SQL : Cours et exercices corrigés [Pratt01]

Un ensemble d'exemples, de questions/réponses, de how-to, et d'exercices corrigés pour travailler les basiques du LMD SQL avec une approche très pratique. Un bon outil pour s'entraîner et un bon compagnon à avoir à ses côtés pour des débuts en SQL (pour la réalisation de travaux pratiques par exemple).

Programmation SQL [Mata03]

De nombreux exemples et exercices, des listings complets, une référence SQL (mots réservés et diagrammes de Conway de la syntaxe).



Complément: Pratique

Comprendre les jointures dans Access [w_mhubiche.developpez.com]

Un tutoriel très pédagogique sur l'expression de jointures sous Access qui aide à comprendre l'opération de jointure en général.

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.

5.1 Redondance et normalisation

Objectifs

Comprendre la problématique de la redondance.

5.1.1 Exercice introductif : Redondance

Soit la relation R suivante, définie en extension :

A	B	C	D	E	F	G
0	1	1	10	5	X	A
0	2	1	10	9	X	G
0	1	2	10	6	X	S
0	1	3	10	7	X	D
1	2	3	20	7	Y	D
0	3	3	10	9	X	G
1	4	3	20	8	Y	F
1	1	4	20	9	Y	G

Tableau 22 Relation R

Question 1

Proposez une clé primaire pour cette relation. Justifiez brièvement.

Question 2

Cette relation contient-elle des redondances ? Si oui lesquelles ? Justifiez brièvement.

Question 3

Si la relation contient des redondances, proposez une solution contenant exactement la même information, mais sans redondance.

5.1.2 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 véhicules 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

Tableau 23 Relation redondante

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).

Complément

On conseillera de lire le chapitre 2 de SQL2 SQL3, applications à Oracle [Delmal01] (pages 42 à 49) qui propose une très bonne démonstration par l'exemple des problèmes posés par une mauvaise modélisation relationnelle.

5.1.3 Principes de la normalisation

Fondamental

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.

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 œuvre de la normalisation est fondée sur la décomposition progressive des relations jusqu'à obtenir des relations normalisées.

5.2 Les dépendances fonctionnelles

Objectifs

Savoir repérer et exprimer des dépendances fonctionnelles.
Définir une clé par les dépendances fonctionnelles.

5.2.1 Dépendance fonctionnelle



Définition : 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 Rssi quelle que soit l'instance r de R, alors pour tous tuples t1 et t2 de r on a :

$$\text{Projection}(t_1, X) = \text{Projection}(t_2, X) \Rightarrow \text{Projection}(t_1, Y) = \text{Projection}(t_2, Y)$$



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 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.

5.2.2 Les axiomes d'Armstrong

Introduction

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



Définition : 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 :

$$XY \rightarrow XY \text{ et } XY \rightarrow X \text{ et } XY \rightarrow Y$$



Définition : 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 :

$$X \rightarrow Y \Rightarrow XZ \rightarrow YZ$$



Définition : Transfert

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 :

$$X \rightarrow Y \text{ et } Y \rightarrow Z \Rightarrow X \rightarrow Z$$

5.2.3 Autres propriétés déduites des axiomes d'Armstrong

Introduction

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



Définition : 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 :

$$X \rightarrow Y \text{ et } WY \rightarrow Z \Rightarrow WX \rightarrow Z$$

Cette propriété est déduite de l'augmentation et de la réflexivité :

$$X \rightarrow Y \text{ et } WY \rightarrow Z \Rightarrow WX \rightarrow WY \text{ et } WY \rightarrow Z \Rightarrow WX \rightarrow Z$$



Définition : Union

Si un attribut détermine plusieurs autres attributs, alors il détermine tout groupe composé de ces attributs.

Soient X, Y et Z des attributs :

$$X \rightarrow Y \text{ et } X \rightarrow Z \Rightarrow X \rightarrow YZ$$

Cette propriété est déduite de la réflexivité, de l'augmentation et de la transitivité :

$$X \rightarrow Y \text{ et } X \rightarrow Z \Rightarrow X \rightarrow XX \text{ et } XX \rightarrow XY \text{ et } YX \rightarrow YZ \Rightarrow X \rightarrow YZ$$



Définition : 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 :

$$X \rightarrow YZ \Rightarrow X \rightarrow Z \text{ et } X \rightarrow Y$$

Cette propriété est déduite de la réflexivité et de la transitivité :

$$X \rightarrow YZ \Rightarrow X \rightarrow YZ \text{ et } YZ \rightarrow Z \Rightarrow X \rightarrow Z$$

5.2.4 DF élémentaire



Définition : Dépendance fonctionnelle élémentaire

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



Exemple : DF élémentaires

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



Exemple : DF non élémentaires

- $AB \rightarrow A$ n'est pas élémentaire car A est inclus dans AB.
- $AB \rightarrow CB$ n'est pas élémentaire car CB n'est pas un attribut, mais un groupe d'attributs.
- $N^{\circ}SS \rightarrow Nom$, Prénom n'est pas élémentaire.



Remarque

On peut toujours réécrire un ensemble de DF en un ensemble de DFE, 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 :

- $AB \rightarrow A$ n'est pas considérée car c'est une DF triviale obtenue par réflexivité.
- $AB \rightarrow CB$ est décomposée en $AB \rightarrow C$ et $AB \rightarrow B$, et $AB \rightarrow B$ n'est plus considérée car triviale.
- $N^{\circ}SS \rightarrow Nom$, Prénom est décomposée en $N^{\circ}SS \rightarrow Nom$ et $N^{\circ}SS \rightarrow Prénom$.

5.2.5 Notion de fermeture transitive des DFE



Définition : 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 $F = \{A \rightarrow B, B \rightarrow C, B \rightarrow D, A \rightarrow E\}$.

La fermeture transitive de F est $F^+ = \{A \rightarrow B, B \rightarrow C, B \rightarrow D, A \rightarrow E, A \rightarrow C, A \rightarrow D\}$

5.2.6 Notion de couverture minimale des DFE



Définition : Couverture minimale

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

Synonymes : 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 $F = \{A \rightarrow B, A \rightarrow C, B \rightarrow C, C \rightarrow B\}$ admet les deux couvertures minimales :

$CM1 = \{A \rightarrow C, B \rightarrow C, C \rightarrow B\}$ et $CM2 = \{A \rightarrow B, B \rightarrow C, C \rightarrow B\}$

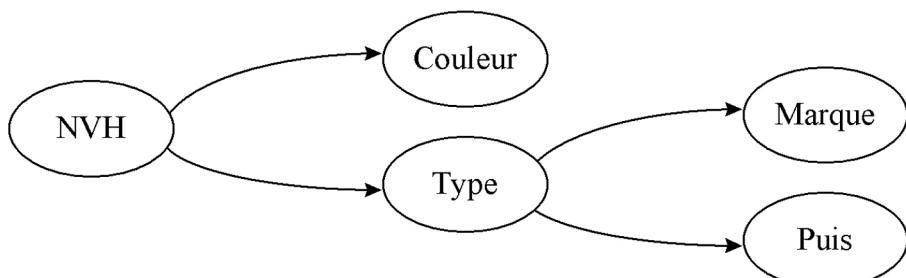
5.2.7 Notion de graphe des DFE

On peut représenter un ensemble de DFE par un graphe orienté (ou plus précisément un réseau de Pétri), tel que les nœuds 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 $F = \{NVH \rightarrow Type, Type \rightarrow Marque, Type \rightarrow Puis, NVH \rightarrow Couleur\}$. On peut représenter F par le graphe ci-dessous :

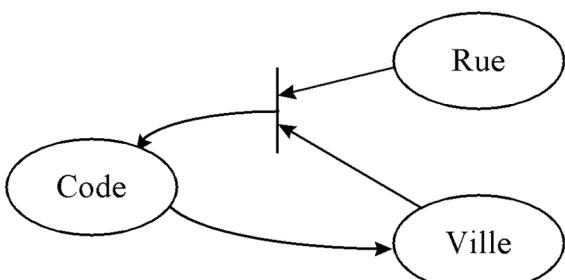


Graphe des DFE de la relation Voiture



Exemple : Relation CodePostal

Soit la relation CodePostal(Code, Ville, Rue) avec l'ensemble des DF $F = \{Code \rightarrow Ville, (Ville, Rue) \rightarrow Code\}$. On peut représenter F par le graphe ci-dessous :



Graphe des DFE de la relation CodePostal

5.2.8 Définition formelle d'une clé



Définition : Clé

Soient une relation $R(A_1, A_2, \dots, A_n)$ et K un sous-ensemble de A_1, A_2, \dots, A_n . K est une clé de R si et seulement si $K \rightarrow A_1, A_2, \dots, A_n$ et il n'existe pas X inclus dans K tel que $X \rightarrow A_1, A_2, \dots, A_n$.

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, chacune est dite clé candidate et l'on en choisit une en particulier pour être la clé primaire.



Attention: Les clés candidates sont des clés !

Toutes les clés candidates sont des clés, pas seulement la clé primaire.



Remarque : Les clés candidates se déterminent mutuellement

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



Complément: Relation "toute clé"

Étant donné qu'une relation dispose forcément d'une clé, si une relation R n'admet aucune clé K sous ensemble des attributs A1..An de R, alors c'est que K=A1..An (la clé est composée de **tous** les attributs de R).

On parle de relation "toute clé".

5.3 Les formes normales

Objectifs

Connaître les **formes normales et leurs implications en terme de redondance**.

5.3.1 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 recenser 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.

5.3.2 Principe de la décomposition



Définition : 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.



Définition : 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).

5.3.3 Première forme normale



Définition : 1NF

Une relation est en 1NF si elle possède au moins une clé et si tous ses attributs sont atomiques.



Définition : 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.



Fondamental : Énoncer les clés

Le modèle relationnel impose qu'une relation ait une clé, donc la condition "est en 1NF si elle possède une clé" est superflue (au pire la relation est "toute clé").

Il est néanmoins fondamental d'avoir identifié les clés au début du processus de normalisation.

5.3.4 Deuxième forme normale

Introduction

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



Définition : 2NF

Une relation est en 2NF 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, Prenom)

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.

5.3.5 Troisième forme normale

Introduction

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

Définition : 3NF

 Une relation est en 3NF 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.

Attention: Clé candidate

 La définition concerne toutes les clés candidates et non uniquement la clé primaireSQL avancé : Programmation et techniques avancées. [Celko00] (p.27).

Exemple : Échelle 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(#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 DFE, issues d'une partie d'une clé, et donc que toutes les DF issues d'une clé sont élémentaires.

Fondamental

 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, 4NF et 5NF) 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.

5.3.6 Forme normale de Boyce-Codd

Introduction

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.



Définition : 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, #Region=>Region, Nom)
Region(#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.



Attention: Non préservation des DF

Une décomposition en BCNF ne préserve en général pas les DF.

5.4 Conception de bases de données normalisées

Objectifs

Savoir créer des schémas relationnels en troisième forme normale.

5.4.1 Processus de conception d'une base de données normalisée

1. Modélisation conceptuelle en UML
Résultat : MCD₁
2. Traduction en relationnel en utilisant les règles de passage UML vers relationnel
Résultat : MLD₁
3. Établissement des DF sous la forme de la fermeture transitive pour chaque relation du modèle
Résultat : MLD₁ avec F+
4. Établissement de la forme normale
Résultat : MLD₁ avec F+ et NF
5. Décomposition des relations jusqu'à arriver en 3NF en préservant les DF
Résultat : MLD₂ avec F+ et 3NF
6. Rétro-conception du modèle UML correspondant
Résultat : MCD₂

5.4.2 Algorithme de décomposition



Méthode : Décomposition ONF->1NF

Soit R une relation avec la clé primaire pk. Si R contient un attribut non atomique, alors R est décomposée en R1 et R2, tel que :

- R1 est R moins l'attribut a
 - R2 est composé de pk et de a, avec (pk,a) clé primaire de R2 et R2.pk clé étrangère vers R1.pk
- R(#pk, a, b, ...) avec a non atomique se décompose en :
- R1(#pk, b, ...)
 - R2(#pk=>R1, #a)



Fondamental : Décomposition > 1NF

Pour les NF supérieures à 1, afin de normaliser une relation R on réalise une décomposition en R1 et R2 pour chaque DFE responsable d'un défaut de normalisation tel que :

- la partie gauche de la DFE :
 - a. devient la clé primaire de R2
 - b. devient une clé étrangère de R1 vers R2
- la partie droite de la DFE
 - a. est enlevée de R1
 - b. est ajoutée comme attributs simples de R2



Méthode : Décomposition 1NF->2NF

Soit R une relation comportant une clé composée de k1 et k1'. Si R contient une DFE de k1' vers des attributs n'appartenant pas à la clé, alors R est décomposée en R1 et R2, tel que :

- R1 est R moins les attributs déterminés par k1' et avec k1' clé étrangère vers R2
- R2 est composée de k1' et des attributs qu'elle détermine, avec k1' clé primaire de R2

R(#pk, k1, k1', a, b, c, ...) avec (k1,K1') clé et k1'>a,b se décompose en :

- R1(#pk, k1, k1'=>R2, c, ...)
- R2(#k1', a, b)



Méthode : Décomposition 2NF->3NF

Soit R une relation comportant une DFE de a vers b qui n'appartiennent pas à une clé, alors R est décomposée en R1 et R2, tel que :

- R1 est R moins les attributs déterminés par a et avec a clé étrangère vers R2
- R2 est composée de a et des attributs qu'elle détermine, avec a clé primaire de R2

R(#pk, a, b, c, ...) avec a>b se décompose en

- R1(#pk, a=>R2, c, ...)
- R2(#a, b)

5.4.3 Exemple de décomposition



Exemple : Situation initiale

Resultat (#pknum, knumetu, kuv, prenom, nom, credits, resultat, obtenu) avec (knumetu, kuv) clé

pknum	knumetu	kuv	prenom	nom	credits	resultat	obtenu
1	X01	NF17	Pierre	Alpha	6	A	oui
2	X01	NF26	Pierre	Alpha	6	B	oui
3	X02	NF17	Alphonse	Béta	6	F	non

- pknum → knumetu,kuv,prenom,nom,credits,resultat,obtenu
- knumetu,kuv → pknum,prenom,nom,credits,resultat,obtenu
- **knumetu → prenom,nom**
- **kuv → credits**
- **resultat → obtenu**

knumetu → prenom,nom

Resultat (#pknum, knumetu=>Etudiant, kuv, credits, resultat, obtenu)

```
Etudiant (#knumetu, prenom, nom)
```

kuv → credits

```
Resultat (#pknum, knumetu=>Etudiant, kuv=>Uv, resultat, obtenu)
Etudiant (#knumetu, prenom, nom)
Uv(#kuv, credits)
```

resultat → obtenu

```
Resultat (#pknum, knumetu=>Etudiant, kuv=>Uv, resultat=>Note)
Etudiant (#knumetu, prenom, nom)
Uv(#kuv, credits)
Note(#resultat, obtenu)
```

* *
*

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.

5.5 Bibliographie commentée sur la normalisation



Complément: Synthèses

SQL2 SQL3, applications à Oracle [Delmal01]

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.

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

Si les SGBD offrent les technologies de modélisation et de gestion des données, ils nécessitent la plupart du temps d'être interfacés avec des applications qui fournissent un accès orienté métier aux utilisateurs, notamment à travers des IHM évoluées. Même des systèmes comme Oracle ou PostgreSQL qui proposent un langage procédural (comme PL/SQL et PL/pgSQL) au dessus de la couche SQL, ne sont pas auto-suffisants. Les langages évolués comme Java ou C sont couramment utilisés pour implémenter la couche applicative d'exploitation des BD.

Les applications de BD sont aujourd'hui généralement réalisées selon des architectures réseaux. L'explosion d'Internet de son côté a favorisé le langage HTML pour implémenter les IHM et a vu la naissance de langages de script pour implémenter la couche applicative côté serveur, tels que PHP, ASP ou JSP.

6.1 Architecture Web

Objectifs

Comprendre les principes des architectures d'application de bases de données (en particulier 3-tier et Web)

Savoir appliquer les principes d'une architecture Web dans le cadre des technologies Servlets ou PHP et HTML

Cette section a été réalisée à partir de contenus de www.commentcamarche.net, © 2003 Jean-François Pillou (document soumis à la licence GNU FDL).

6.1.1 Notions d'architecture client-serveur

Présentation de l'architecture d'un système client/serveur

De nombreuses applications fonctionnent selon un environnement clients/serveur, cela signifie que des machines clientes (des machines faisant partie du réseau) contactent un serveur, une machine généralement très puissante en terme de capacités d'entrée-sortie, qui leur fournit des services. Ces services sont des programmes fournissant des données telles que l'heure, des fichiers, une connexion...

Les services sont exploités par des programmes, appelés programmes clients, s'exécutant sur les machines clientes. On parle ainsi de client FTP, client de messagerie...

Dans un environnement purement client/serveur, les ordinateurs du réseau (les clients) ne peuvent voir que le serveur, c'est un des principaux atouts de ce modèle.

Avantages de l'architecture client/serveur

Le modèle client/serveur est particulièrement recommandé pour des réseaux nécessitant un grand niveau de fiabilité, ses principaux atouts sont :

- **des ressources centralisées**
étant donné que le serveur est au centre du réseau, il peut gérer des ressources communes à tous les utilisateurs, comme par exemple une base de données centralisée, afin d'éviter les problèmes de redondance et de contradiction
- **une meilleure sécurité**
car le nombre de points d'entrée permettant l'accès aux données est moins important
- **une administration au niveau serveur**
les clients ayant peu d'importance dans ce modèle, ils ont moins besoin d'être administrés
- **un réseau évolutif**
grâce à cette architecture ont peu supprimer ou rajouter des clients sans perturber le fonctionnement du réseau et sans modifications majeures

Inconvénients du modèle client/serveur

L'architecture client/serveur a tout de même quelques lacunes parmi lesquelles :

- **un coût élevé**
dû à la technicité du serveur
- **un maillon faible**
le serveur est le seul maillon faible du réseau client/serveur, étant donné que tout le réseau est architecturé autour de lui! Heureusement, le serveur a une grande tolérance aux pannes (notamment grâce au système RAID)

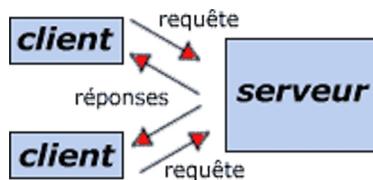
Fonctionnement d'un système client/serveur

Schéma de fonctionnement d'un système client/serveur (commentcamarche.net - © 2003 Pillou - GNU FDL)

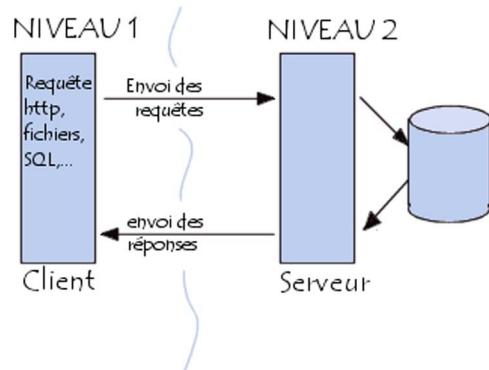
Un système client/serveur fonctionne selon le schéma suivant:

- Le client émet une requête vers le serveur grâce à son adresse et le port, qui désigne un service particulier du serveur
- Le serveur reçoit la demande et répond à l'aide de l'adresse de la machine client et son port

6.1.2 Notions d'architecture 3-tier

Présentation de l'architecture à deux niveaux

L'architecture à deux niveaux (aussi appelée architecture 2-tier, *tier* signifiant étage en anglais) caractérise les systèmes clients/serveurs dans lesquels le client demande une ressource et le serveur la lui fournit directement. Cela signifie que le serveur ne fait pas appel à une autre application afin de fournir le service.

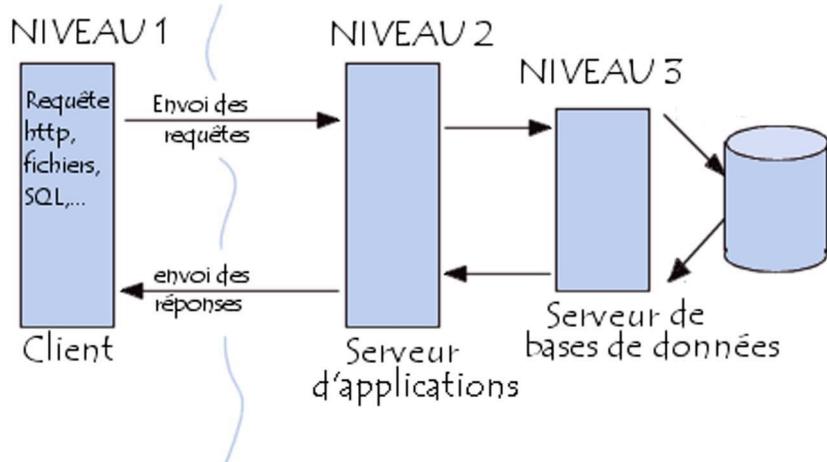


Architecture 2-tier (commentcamarche.net - © 2003 Pillou - GNU FDL)

Présentation de l'architecture à trois niveaux

Dans l'architecture à 3 niveaux (appelée architecture 3-tier), il existe un niveau intermédiaire, c'est-à-dire que l'on a généralement une architecture partagée entre:

1. **Le client**
le demandeur de ressources
2. **Le serveur d'application**
(appelé aussi *middleware*) le serveur chargé de fournir la ressource mais faisant appel à un autre serveur
3. **Le serveur secondaire**
(généralement un serveur de base de données), fournissant un service au premier serveur



Architecture 3-tier (commentcamarche.net - © 2003 Pillou - GNU FDL)

Remarque

Étant donné l'emploi massif du terme d'architecture à 3 niveaux, celui-ci peut parfois désigner aussi les architectures suivantes :

- Partage d'application entre client, serveur intermédiaire, et serveur d'entreprise
- Partage d'application entre client, base de données intermédiaire, et base de données d'entreprise

Comparaison des deux types d'architecture

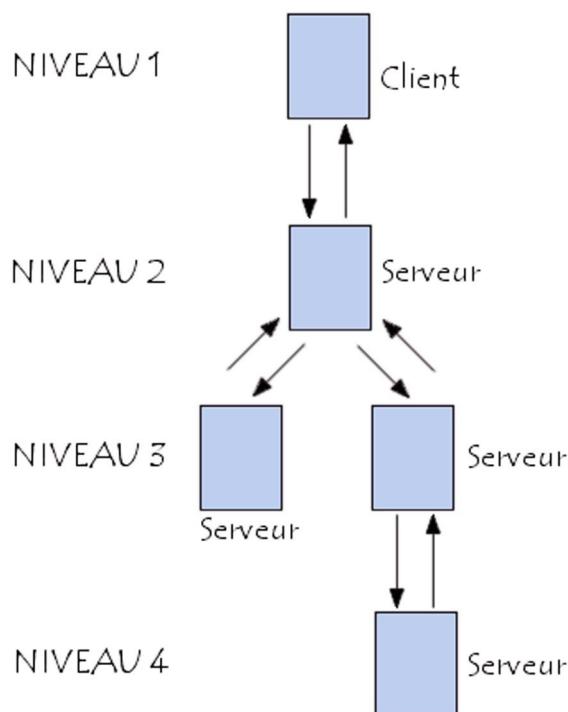
L'architecture à deux niveaux est donc une architecture client/serveur dans laquelle le serveur est polyvalent, c'est-à-dire qu'il est capable de fournir directement l'ensemble des ressources demandées par le client.

Dans l'architecture à trois niveaux par contre, les applications au niveau serveur sont délocalisées, c'est-à-dire que chaque serveur est spécialisé dans une tâche (serveur web et serveur de base de données par exemple). Ainsi, l'architecture à trois niveaux permet :

- une plus grande flexibilité/souplesse
- une plus grande sécurité (la sécurité peut être définie pour chaque service)
- de meilleures performances (les tâches sont partagées)

Complément: L'architecture multi-niveaux

Dans l'architecture à 3 niveaux, chaque serveur (niveaux 1 et 2) effectue une tâche (un service) spécialisée. Ainsi, un serveur peut utiliser les services d'un ou plusieurs autres serveurs afin de fournir son propre service. Par conséquence, l'architecture à trois niveaux est potentiellement une architecture à N niveaux.



Architecture N-tier (commentcamarche.net - © 2003 Pillou - GNU FDL)

6.1.3 Notions de serveur Web

Fait à partir de www.commentcamarche.net. Copyright 2003 Jean-François Pillou. Document soumis à la licence GNU FDL.

Un serveur web est un logiciel permettant à des clients d'accéder à des pages web, c'est-à-dire en réalité des fichiers au format HTML à partir d'un navigateur (aussi appelé browser) installé sur leur ordinateur distant.

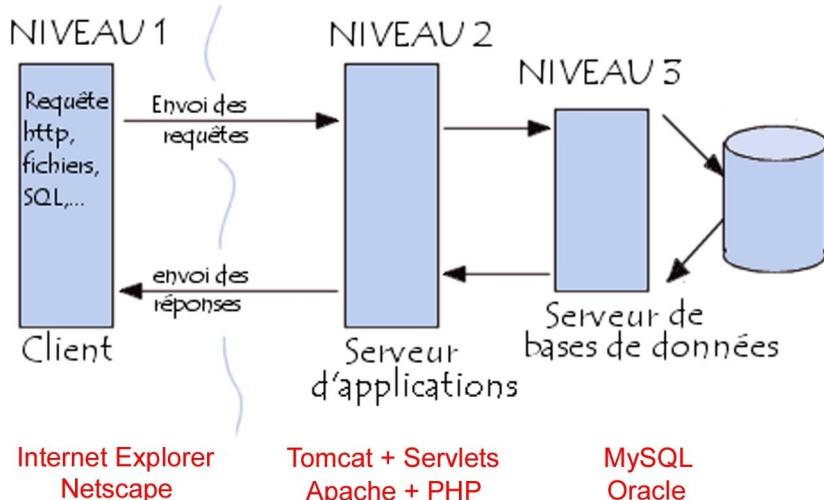
Un serveur web est donc un "simple" logiciel capable d'interpréter les requêtes HTTP arrivant sur le port associé au protocole HTTP (par défaut le port 80), et de fournir une réponse avec ce même protocole.

Les principaux serveurs web sur le marché sont entre autres :

- Apache
- Microsoft IIS (Internet Information Server)
- Microsoft PWS (Personal Web Server)
- ...

6.1.4 Notion d'architecture Web

Fait à partir de www.commentcamarche.net. Copyright 2003 Jean-François Pillou. Document soumis à la licence GNU FDL.



Exemples d'architecture Web (commentcamarche.net - © 2003 Pillou - GNU FDL)

6.1.5 Architecture LAAP



Définition : Définition

On appelle une architecture LAPP une architecture qui s'appuie sur :

- Linux pour le système d'exploitation
- Apache pour le serveur Web
- PostgreSQL pour la base de données
- PHP pour le langage applicatif



Complément: LAMP, WAMP, WAPP

- LAMP : Linux, Apache, MySQL, PHP
- WAMP : Windows, Apache, MySQL, PHP
- ...

6.2 Introduction à HTML

Objectifs

Savoir écrire une page simple en HTML

Savoir créer des formulaires en HTML

6.2.1 HTML



Définition : HTML

HTML est un langage inventé à partir de 1989 pour coder des pages de contenu sur le Web. Il est standardisé par le W3C.



Définition : Langage à balises

HTML est un langage à balises : il se fonde sur le mélange entre du contenu et des balises permettant de caractériser ce contenu. HTML utilise le formalisme SGML pour définir les balises et combinaisons de balises autorisées.



Exemple : Extrait de code HTML

```
<p>Ceci est un contenu, caractérisé par des <b>balises</b></p>
```

Les balises p et b ont une signification dans le langage HTML : Créer un paragraphe et mettre en gras.

**Remarque : HTML5**

La version courante de HTML et la version 4.01 de 1999. Le HTML5 en cours de spécification est déjà dans un état avancé de spécification et d'implémentation, il peut d'ors et déjà être employé et est prévu pour 2014.

6.2.2 XHTML

**Définition : XHTML**

XHTML est une réécriture du HTML : tandis que HTML est fondé sur SGML, XHTML est fondé sur XML, plus récent et plus rigoureux. XHTML et HTML ne présentent pas de différence fonctionnelle, uniquement des différences syntaxiques.

**Exemple : Comparaison XHTML et HTML**

```
<ul><li>Ceci est un extrait de contenu <i>HTML
```

```
<ul><li>Ceci est un extrait de contenu <i>XHTML</i></li></ul>
```

Dans le cas du HTML les balises fermantes sont optionnelles, en XHTML c'est obligatoire. Les deux exemples sont donc équivalents, mais dans l'exemple HTML, il existait en fait plusieurs interprétations possibles, par exemple :

```
<ul><li>Ceci est un extrait de contenu</li></ul><i>XHTML</i>
```

**Remarque : XHTML5**

La version actuelle de XHTML est la version 1, correspondant à HTML4. XHTML5 est le pendant de HTML5.

**Complément**

Définition du XML (cf. *Définition du XML*)

Historique : de SGML à XML (cf. *Historique : de SGML à XML*)

Discussion : HTML et XML (cf. *Discussion : HTML et XML*)

6.2.3 Structure générale XHTML

**Syntaxe : Structure générale**

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    ...
  </head>
  <body>
    ...
  </body>
</html>
```

**Syntaxe : Entête**

```
<head>
  <title>...</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
```

**Syntaxe : Corps**

```
<body>
  <h1>...</h1>
  <h2>...</h2>
  <p>...</p>
</body>
```

**Complément**

- Tutoriel XHTML : <http://fr.html.net/tutorials/html/>
- Brillant07, pp107-108 [Brillant07]

6.2.4 Balises de base XHTML



Syntaxe

```
<p>Un paragraphe de texte</p>
<p>Paragraphe contenant du texte, mot <b>gras</b> ou <i>italique</i>. </p>
<p><a href="page02.html">Un lien</a> vers une autre page</p>

<h1>Titre de niveau 1</h1>
<h2>Titre de niveau 2</h2>
<h3>Titre de niveau 3</h3>
<table border="1">
  <tr><th>Titre colonne 1</th><th>Titre colonne 2</th><th>...</th></tr>
  <tr><td>Ligne 1 colonne 1</td><td>Ligne 1 colonne 2</td><td>...</td></tr>
  <tr><td>Ligne 2 colonne 1</td><td>Ligne 2 colonne 2</td><td>...</td></tr>
</table>
<ul>
  <li>Item de liste à puces</li>
  <li>Item de liste à puces</li>
</ul>
<ol>
  <li>Item de liste à ordonnée</li>
  <li>Item de liste à ordonnée</li>
</ol>
```



Complément

Pour une description des balises de base : Brillant07, pp108-112 [Brillant07].

6.2.5 Introduction à CSS

CSS (Cascading Style Sheets) est un standard du W3C qui complète HTML.

CSS sert à :

- Mieux séparer méthodologiquement la structure (en HTML) de la mise en forme (CSS)
- Simplifier l'écriture HTML (et la génération HTML dans le cas de sites dynamiques réalisés avec PHP par exemple)
- Factoriser et réutiliser la mise en forme, notamment pour assurer l'homogénéisation d'un site Web (externalisation des feuilles CSS)
- Augmenter les possibilités de mise en forme du HTML (couches CSS)



Syntaxe : Association HTML et CSS

On peut associer à tout fichier HTML, dans le head, une ou plusieurs feuilles CSS (qui se complètent "**en cascade**").

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>...</title>
  <link href='css/style1.css' type='text/css' rel='stylesheet'/>
  <link href='css/style2.css' type='text/css' rel='stylesheet'/>
  ...
...
```



Syntaxe : Syntaxe CSS

Un fichier CSS permet d'associer une mise en forme aux éléments utilisés dans le fichier HTML.

```
p { font-family:sans-serif; }
```



Méthode : DIV et SPAN

On peut différencier les styles en fonction des attributs class et id des éléments HTML.

On recourt alors massivement aux éléments div et span pour le stylage, avec des attributs id (pour référencer un élément particulier) et/ou class (pour référencer un ensemble d'éléments de même type).

```
div.class { ... }
span.class { ... }
#id { ... }
```



Complément

Tutoriel CSS didactique et complet : <http://fr.html.net/tutorials/css/>.

Pour une introduction à CSS : Brillant07, pp112-122 [Brillant07].

Zen Garden : <http://www.csszengarden.com>³ ; Le Zen des CSS [Shea06]

6.2.6 Mise en ligne

- ~/public_html
- donner les droits en lecture sur les fichiers (chmod 755)
- le fichier est ensuite accessible via une adresse telle que :
<http://tuxa.sme.utc/~nf17pXXX/monfichier.html>

6.2.7 Formulaires HTML

Fait à partir de www.commentcamarche.net. Copyright 2003 Jean-François Pillou. Document soumis à la licence GNU FDL.

Grâce à la balise FORM du langage HTML, il est très simple de créer des formulaires comprenant :

- des cases à cocher
- des champs de saisie
- des boutons radio
- des listes à choix multiples
- ...

Exemple : Formulaire

```
<form Method="GET" Action="test.php">
    Nom : <input type="text" name="nom"><br/>
    Prénom : <input type="text" name="prenom"><br/>
    Age : <input type="text" name="age"><br/>
    <input type="submit">
</form>
```

Exemple de formulaire HTML

6.3 Introduction à PHP

Objectifs

- Comprendre le fonctionnement d'un langage applicatif côté serveur comme PHP
- Savoir faire des programmes simples en PHP
- Connaître les possibilités objet de PHP

Cette section a été réalisée à partir de contenus de www.commentcamarche.net, © 2003 Jean-François Pillou (document soumis à

3 - <http://www.csszengarden.com/>

la licence GNU FDL).

6.3.1 Présentation de PHP

PHP est un langage interprété (un langage de script) exécuté du côté serveur (comme les scripts CGI, ASP, ...) et non du côté client (un script écrit en JavaScript ou une applet Java s'exécute au contraire sur l'ordinateur où se trouve le navigateur). La syntaxe du langage provient de celles du langage C, du Perl et de Java.

Ses principaux atouts sont :

- La gratuité et la disponibilité du code source (PHP est distribué sous licence GNU GPL)
- La simplicité d'écriture de scripts
- La possibilité d'inclure le script PHP au sein d'une page HTML (contrairement aux scripts CGI, pour lesquels il faut écrire des lignes de code pour afficher chaque ligne en langage HTML)
- La simplicité d'interfaçage avec des bases de données (de nombreux SGBD sont supportés, le plus utilisé avec ce langage est MySQL).
- L'intégration au sein de nombreux serveurs web (Apache, Microsoft IIS, ...)

 Exemple : SGBD supportés par PHP

- MySQL
- Oracle
- PostgreSQL
- ...

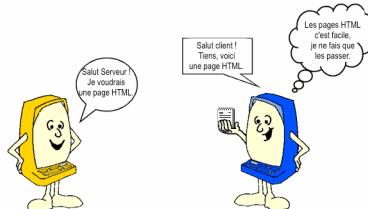
6.3.2 Principes de PHP

L'interprétation du code par le serveur

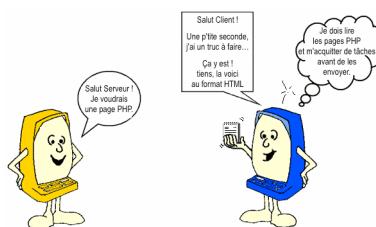
Un script PHP est un simple fichier texte contenant des instructions écrites à l'aide de caractères ASCII 7 bits (des caractères non accentués) incluses dans un code HTML à l'aide de balises spéciales et stocké sur le serveur. Ce fichier doit avoir une extension particulière (qui dépend de la configuration du serveur HTTP, en général ".php") pour pouvoir être interprété par le serveur.

Ainsi, lorsqu'un navigateur (le client) désire accéder à une page dynamique réalisée en php :

1. Le serveur reconnaît qu'il s'agit d'un fichier PHP
2. Il lit le fichier PHP
3. Dès que le serveur rencontre une balise indiquant que les lignes suivantes sont du code PHP, il "passe" en mode PHP, ce qui signifie qu'il ne lit plus les instructions: il les exécute.
4. Lorsque le serveur rencontre une instruction, il la transmet à l'interpréteur
5. L'interpréteur exécute l'instruction puis envoie les sorties éventuelles au serveur
6. A la fin du script, le serveur transmet le résultat au client (le navigateur)



Requête d'une page HTML (<http://fr.html.net/tutorials/php/lesson1.php>)



Requête d'une page PHP (<http://fr.html.net/tutorials/php/lesson1.php>)

 Remarque : Code PHP et clients Web

Un script PHP est interprété par le serveur, les utilisateurs ne peuvent donc pas voir le code source !

Le code PHP stocké sur le serveur n'est donc jamais visible directement par le client puisque dès qu'il en demande l'accès, le serveur l'interprète ! De cette façon aucune modification n'est à apporter sur les navigateurs...

Implantation au sein du code HTML

Pour que le script soit interprété par le serveur deux conditions sont nécessaires :

- Le fichier contenant le code doit avoir l'extension .php et non .html (selon la configuration du serveur Web)
- Le code PHP contenu dans le code HTML doit être délimité par les balises "<?php" et "?>"

 Exemple : Hello world

```
<html>
<head><title>Exemple</title></head>
<body>
<?php
echo "Hello world";
?>
</body>
</html>
```

Complément

 Tutoriel PHP : <http://fr.html.net/tutorials/php/>

6.3.3 Syntaxe PHP

 Fondamental : Manuel PHP en ligne

<http://php.net/manual/>

 Exemple

```
<?php
$i=0 ;
while($i<6) {
    echo $i ;
    $i=rand(1,6) ;
}
```

Attention: Généralités

- 
- Une instruction se termine par un ;
 - Les espaces, retours chariot et tabulation ne sont pas pris en compte par l'interpréteur
 - Les commentaires sont écrits entre les délimiteurs /* et */ ou // sur une seule ligne.
 - Le langage est case-sensitive (sauf pour les fonctions).

Complément: IDE

- 
- Eclipse PDT (PHP Development Tools)
<http://www zend com/fr/community/pdt>
 - Zend Studio
<http://www zend com/fr/products/studio/>

6.3.4 Variables en PHP

- Les variables ne sont pas déclarées
- Les variables commencent pas un \$
- Les variables ne sont pas typées

Les variables en langage PHP peuvent être de trois types :

- Scalaires (entiers, chaîne, réels)
- Tableaux (un tableau pouvant être multidimensionnel et stocker des scalaires de types différents)
- Tableaux associatifs (indexés par des chaînes)

 Exemple

```
$Entier=1;
$Reel=1.0;
$Chaine="1";
$Tableau[0]=1
```

```
$Tableau[1]="1"
$TableauMulti[0][0]="1.0"
$TableauAssoc[Age]=18
```



Complément: isset()

```
if (isset($var)) {
    echo $var;
}
```

La fonction `isset()` permet de tester qu'une variable existe et est affectée.

6.3.5 Structures de contrôle en PHP

```
if (condition réalisée) {
    liste d'instructions
}
elseif (autre condition réalisée) {
    autre série d'instructions
}
...
else (dernière condition réalisée) {
    série d'instructions
}
```



Complément: Switch

`switch / case / default / break`

6.3.6 Boucles en PHP

```
for (compteur; condition; modification du compteur) {
    liste d'instructions
}
```

```
while (condition réalisée) {
    liste d'instructions
}
```

6.3.7 Fonctions en PHP

```
function Nom_De_La_Fonction(argument1, argument2, ...) {
    liste d'instructions
    ...
    return valeur_ou_variable;
    ...
}
```

6.3.8 Objets en PHP



Syntaxe : Déclaration d'une classe

```
class NomClasse {
    // Déclarations des attributs
    public $donneeMembre1;
    public $donneeMembre2;
    ...
    // Déclarations du constructeur
    function __construct () {
        ...
    }
    // Déclarations des méthodes
```

```
public function Nom_de_la_fonction_membre1(parametres) {
    ...
}
```



Remarque : This

Le mot clé `$this` permet d'accéder à l'objet en cours lors de la déclaration des méthodes.



Syntaxe : Instanciation d'objets

```
$Nom_de_l_objet = new Nom_de_la_classe;
```



Syntaxe : Accès aux propriétés

```
$Nom_de_l_objet->Nom_de_la_propriété = Valeur;
```



Syntaxe : Accès aux méthodes

```
$Nom_de_l_objet->Nom_de_la_méthode (parametre1,parametre2,...);
```



Exemple : Classe de connexion à une base de données PostgreSQL

```
<?php
class Connect {
    var $fHost;
    var $fPort;
    var $fDbname;
    var $fUser;
    var $fPassword;
    var $fConn;
    function __construct () {
        $this->fHost="foo.fr";
        $this->fPort="5432";
        $this->fDbname="myDb";
        $this->fUser="Me";
        $this->fPassword="Secret";
    }
    function mConnect () {
        $this->fConn = pg_connect("host=$this->fHost port=$this->fPort dbname=$this-
>fDbname user=$this->fUser password=$this->fPassword") or die('Échec de la connexion :
'. pg_last_error());
    }
    function mClose () {
        pg_close($this->fConn);
    }
}
?>
```



Exemple : Utilisation de la classe de connexion

```
<?php
    include "connect_class.php";
    $vConnect = new Connect;
    $vConnect->mConnect();
?>
```

6.3.9 Envoi de texte au navigateur



Syntaxe

```
echo Expression;
```



Remarque : Print

La fonction print est iso-fonctionnelle avec echo et printf plus complexe permet en plus le formatage des données (peu utilisée).



Attention: L'importance de l'implantation du code PHP au sein du code HTML

Le code PHP peut être implanté au sein du code HTML. Cette caractéristique n'est pas à négliger car le fait d'écrire uniquement du code PHP là où il est nécessaire rend la programmation plus simple (il est plus simple d'écrire du code HTML que des fonctions echo ou print, dans lesquelles les caractères spéciaux doivent être précédés d'un antislash sous peine de voir des erreurs lors de l'exécution).

L'exemple le plus simple concerne les pages dynamiques dont l'en-tête est toujours le même: dans ce cas, le code PHP peut ne commencer qu'à partir de la balise <body>, au moment où la page peut s'afficher différemment selon une variable par exemple.

Mieux, il est possible d'écrire plusieurs portions de script en PHP, séparées par du code HTML statique car les variables/fonctions déclarées dans une portion de script seront accessibles dans les portions de scripts inférieures.

6.3.10 Formulaires HTML et PHP

PHP rend très simple la récupération de données envoyées par l'intermédiaire de formulaires HTML.

Lorsque l'on soumet un formulaire à un fichier PHP, toutes les données du formulaire lui sont passées en tant que variables, c'est-à-dire chacun des noms associés aux champs (ou boutons) du formulaire précédés du caractère \$.



Exemple : Page d'appel

```
<html>
<body>
  <form method="GET" action="test.php">
    <input type="text" size="20" name="MaVar"/>
    <input type="submit"/>
  </form>
</body>
</html>
```



Exemple : Page appelée (test.php)

```
<?php
echo $MaVar;
?>
```



Remarque : \$HTTP_GET_VARS[] et \$HTTP_POST_VARS[]

Selon la configuration du module PHP, il est possible que la récupération directe des données issue du formulaire HTML ne fonctionne pas. On peut dans ce cas utiliser les tableaux associatifs \$HTTP_GET_VARS['variable'] et \$HTTP_POST_VARS['variable'] ou \$_GET['variable'] et \$_POST['variable'] (selon configuration).

```
<?php
$MaVarLocale=$HTTP_GET_VARS['MaVar']
echo $MaVarLocale;
?>
```



Remarque : Code implanté

La page renvoyée dans l'exemple n'est pas du HTML (mais du simple texte qu'un navigateur peut néanmoins afficher). Pour retourner du HTML, il faut planter ce même code au sein d'une page HTML.

```
<html>
<body>
  <?php
    echo $MaVar;
  ?>
</body>
</html>
```



Attention: Cache

Les navigateurs disposent d'un cache, c'est à dire d'une copie locale des fichiers qui leur évite de recharger plusieurs fois un fichier identique. Lorsque l'on développe une application PHP, les fichiers changent fréquemment, il est alors nécessaire de **vider le cache** pour que le navigateur recharge bien la nouvelle version du code.

Sous Firefox, faire CTRL+F5.

6.3.11 Variables de session



Définition : Variable de session PHP

Une variable de session PHP est une variable stockée sur le serveur.

C'est une variable temporaire qui a une durée limitée et est détruite à la déconnexion (fermeture du navigateur).

Les variables de session sont **partagées** par toutes les pages PHP d'une session (accès depuis un même navigateur). Elles permettent donc le passage d'information entre pages.



Exemple : Exemple d'utilisation

- Sauvegarde d'identifiants de connexion
- Sauvegarde d'un panier d'achat
- ...



Syntaxe : Démarrer une session

```
<?php session_start(); ?>
```

Ce code permet de charger le fichier contenant les variables de session sur le serveur, ou s'il n'existe pas de la créer.



Remarque

Ce code est à placer au début de toutes les pages PHP qui souhaitent utiliser les variables de sessions, avant tout autre code PHP ou HTML.



Syntaxe : Utiliser les variables

```
<?php
...
$_SESSION['variable'] = valeur ;
...
?>
```

Un tableau association `$_SESSION` est alors mis à disposition pour gérer des variables.



Exemple

```
<?php
// page1.php
session_start();
?>
<html>
<body>
<h1>Page 1</h1>
<?php
$_SESSION['login'] = 'me';
$_SESSION['mdp'] = 'secret';
?>
<a href="page2.php">page 2</a>
</body>
</html>
```

```
<?php
// page2.php
session_start();
?>
<html>
<body>
<h1>Page 2</h1>
<?php
echo $_SESSION['login'] ;
echo "<br/>" ;
echo $_SESSION['mdp'] ;
?>
</body>
</html>
```

La page `page2.php` est en mesure d'afficher les informations de la page `page1.php`.



Syntaxe : Autres instructions

- Supprimer une variable : `unset($_SESSION['variable'])`
- Supprimer toutes les variables : `session_unset()`

- Supprimer la session : `session_destroy()`



Complément: Sources

<http://www.phpsources.org/tutoriel-SESSIONS.htm>
<http://www.php.net/manual/fr/book.session.php>



Complément: Cookies

Les sessions s'appuient sur les cookies, fichiers de données gérés côté client par le navigateur Web, pour stocker l'identifiant de session. Il est possible d'utiliser des sessions sans cookie, en passant l'identifiant de session dans l'URL.

<http://www.phpsources.org/tutoriel-cookies.htm>

6.4 Introduction à PostgreSQL

6.4.1 Présentation

PostgreSQL est :

- un SGBDRO
- libre (licence BSD)
- multi-plateforme (Unix, Linux, Windows, MacOS, ...)
- puissant (proche d'Oracle)
- très respectueux du standard



Complément

- <http://www.postgresql.org/>
- <http://www.postgresql.fr/>

6.4.2 Types de données

Types standards

- numériques : integer (int2, int4, int8), real (float4, float8)
- dates : date (time, timestamp)
- chaînes : char, varchar, text
- autres : boolean, array[]



Complément: Documentation

<http://docs.postgresqlfr.org/8.1/datatype.html>



Complément: OID

Voir *Identification d'objets et références* (cf. Identification d'objets et références)



Complément: Types composites

Voir *Les types utilisateurs* (cf. Les types utilisateurs)

6.4.3 Le client textuel "psql"



Définition : psql

psql est le client textuel de PostgreSQL.

```
psql dbname
```



Syntaxe : Commande de base

- \? : Liste des commandes psql
- \h : Liste des instructions SQL
- \h CREATE TABLE : Description de l'instruction SQL CREATE TABLE
- \q : Quitter psql

- \d : Liste des relations (catalogue de données)
- \d maTable : Description de la relation maTable
- \H : mode HTML ou mode textuel pour les retours de requête

Syntaxe : Écrire une instruction SQL

Une instruction SQL peut s'écrire sur une ou plusieurs lignes, le retour chariot n'a pas d'incidence sur la requête, c'est le ; qui marque la fin de l'instruction SQL et provoque son exécution.

```
dbnf17p015=> SELECT * FROM maTable ;
```

```
dbnf17p015=> SELECT *
dbnf17p015-> FROM maTable
dbnf17p015-> ;
```

On notera dans psql la différence entre les caractères => et -> selon que l'on a ou pas effectué un retour chariot.

6.4.4 Exécuter un fichier SQL

Il est souvent intéressant d'exécuter un fichier contenant une liste de commandes SQL, plutôt que de les entrer une par une dans le terminal. Cela permet en particulier de recréer une base de données à partir du script de création des tables.

Syntaxe

Pour exécuter un fichier contenant du code SQL utiliser la commande PostgreSQL \i chemin/fichier.sql

- chemin désigne le répertoire dans lequel est le fichier fichier.sql
- le dossier de travail de psql est le dossier dans lequel il a été lancé, le script peut être lancé à partir de son dossier home pour en être indépendant (~/.psql/fichier.sql)

6.4.5 Fichier CSV

Définition : Fichier CSV

CSV est un format informatique permettant de stocker des données tabulaires dans un fichier texte.

Chaque ligne du fichier correspond à une ligne du tableau. Les valeurs de chaque colonne du tableau sont séparées par un caractère de séparation, en général une **virgule** ou un **point-virgule**. Chaque ligne est terminée par un **caractère de fin de ligne** (*line break*).

Toutes les lignes contiennent **obligatoirement** le même nombre de valeurs (donc le même nombre de caractères de séparation). Les valeurs vides doivent être exprimées par deux caractères de séparation contigus.

La taille du tableau est le nombre de lignes multiplié par le nombre de valeurs dans une ligne.

La première ligne du fichier peut être utilisée pour exprimer le nom des colonnes.

Syntaxe

```
[NomColonne1;NomColonne2;...;NomColonneN]
ValeurColonne1;ValeurColonne2;...;ValeurColonneN
ValeurColonne1;ValeurColonne2;...;ValeurColonneN
...
```

Exemple : Fichier CSV sans entête

```
Pierre;Dupont;20;UTC;NF17
Pierre;Dupont;20;UTC;NF26
Paul;Durand;21;UTC;NF17
Jacques;Dumoulin;21;UTC;NF29
```

Exemple : Fichier CSV avec entête

```
Prenom;Nom;Age;Ecole;UV
Pierre;Dupont;20;UTC;NF17
Pierre;Dupont;20;UTC;NF26
Paul;Durand;21;UTC;NF17
Jacques;Dumoulin;21;UTC;NF29
```





Exemple : Valeur nulle

Jacques;Dumoulin;;UTC;NF29

L'âge est inconnu (NULL).



Attention: Variations...

La syntaxe des fichiers CSV n'est pas complètement standardisée, aussi des variations peuvent exister :

- Les chaînes de caractères peuvent être protégées par des guillemets (les guillemets s'expriment alors avec un double guillemet).
- Le caractère de séparation des nombres décimaux peut être le point ou la virgule (si c'est la virgule, le caractère de séparation doit être différent)
- ...

Un des problèmes les plus importants reste l'encodage des caractères qui n'est pas spécifié dans le fichier et peut donc être source de problèmes, lors de changement d'OS typiquement.



Méthode : Usage en base de données

Les fichiers CSV sont très utilisés en BD pour échanger les données d'une table (export/import).

Les SGBD contiennent généralement des utilitaires permettant d'exporter une table ou un résultat de requête sous la forme d'un fichier CSV, en spécifiant un certain nombre de paramètres (caractère de séparation de valeur, caractère de fin de ligne, présence ou non d'une ligne de définition des noms des colonnes, etc.). De même ils proposent des utilitaires permettant d'importer un fichier CSV dans une table (en spécifiant les mêmes paramètres), voire de créer directement une table à partir du fichier CSV (quand les noms des colonnes sont présents).



Complément: Fichiers à largeur de colonne fixe

Les fichiers à largeur de colonne fixe n'utilisent pas de séparateur de colonne, mais imposent **le même nombre de caractères** pour chaque cellule. L'avantage est de ne pas avoir à spécifier le caractère de séparation, l'inconvénient est la taille de fichier supérieure si les valeurs ne font pas toutes la même largeur.



Complément: XML

Les fichiers XML tendent de plus en plus à remplacer les fichiers CSV car ils permettent d'être beaucoup plus expressifs sur le schéma d'origine. Ils sont également plus standards (encodage spécifié, principe de séparation des données par les tags, etc.). Leur seul inconvénient est d'être plus verbeux et donc plus volumineux.



Complément: Tables externes

Certains SGBD, comme Oracle, permettent de créer des tables dites **externes**, qui autorisent de créer un schéma de table **directement sur un fichier CSV**, permettant ainsi un accès SQL standard à un fichier CSV, sans nécessité de l'importer d'abord dans une table.

6.4.6 Importer un fichier CSV

```
\copy nom_table (att1, att2, ...) FROM 'fichier.csv' WITH DELIMITER AS ',';
```



Remarque

- La table nom_table doit déjà exister
- Le nombre de colonnes spécifié doit correspondre au nombre de colonnes du fichier CSV
- Les types doivent être compatibles



Remarque

Ajouter WITH CSV HEADER si le fichier CSV contient une ligne d'entête.

6.4.7 Le client graphique "pgAdminIII"

Le client graphique pgAdminIII est plus convivial que psql. Néanmoins s'il fonctionne très bien sous Linux, il est instable sous Windows.

Déclarer une connexion

1. Sélectionner Fichier > Ajouter un serveur
2. Utilisez votre compte et mot de passe NF17
 - Hôte : tuxa.sme.utc
 - Port : 5432 (port standard de PostgreSQL)
 - Base de données : dbnf17...

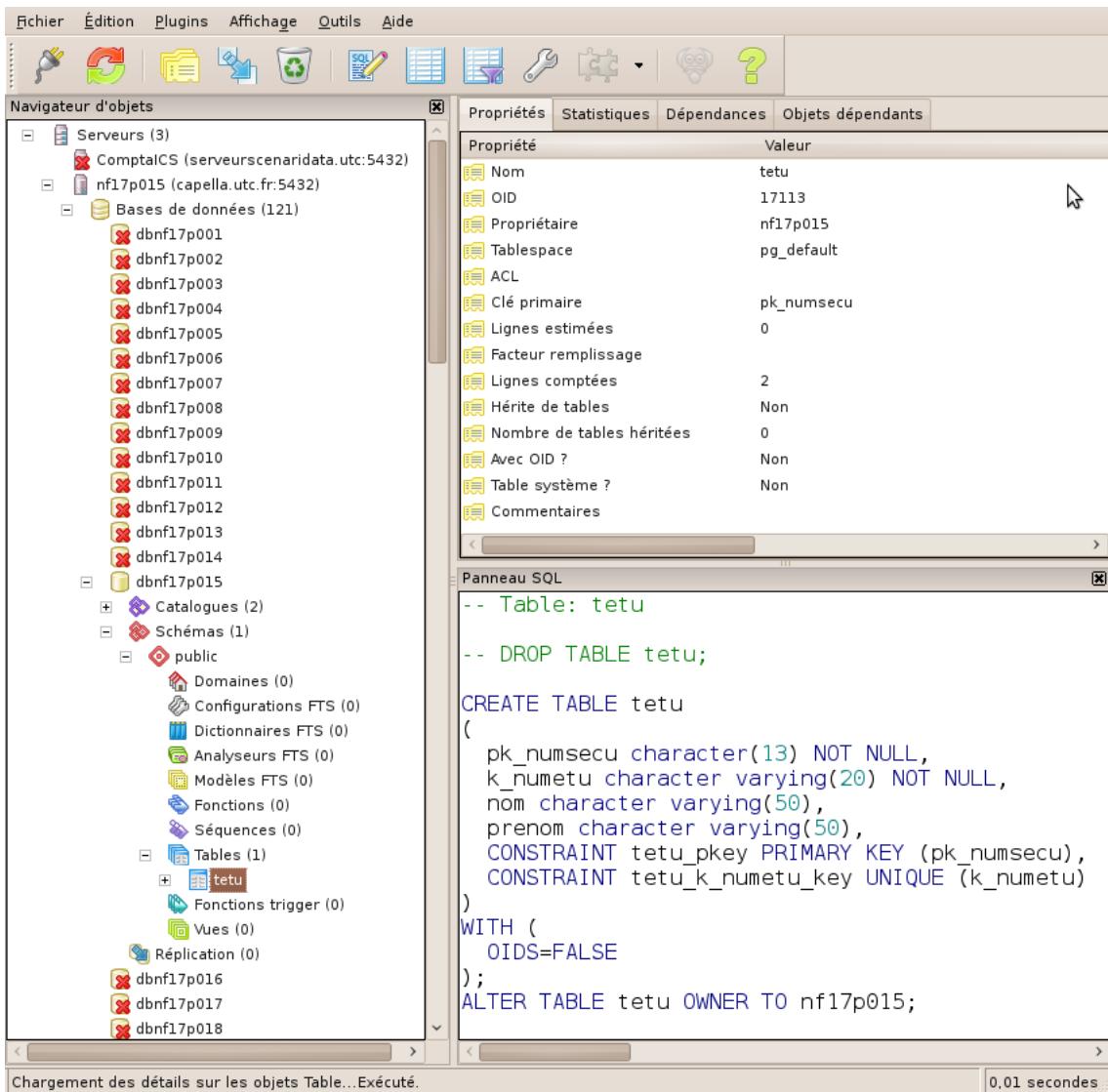
- Utilisateur : nf17...
- Mot de passe : ...



Fenêtre de connexion pgAdmin III

Ouvrir un terminal SQL

1. Sélectionner sa base de données dans la liste Bases de données
2. Sélectionner Outils > Éditeur de requêtes (ou CTRL+E)



pgAdminIII

6.4.8 Notion de schéma

Définition

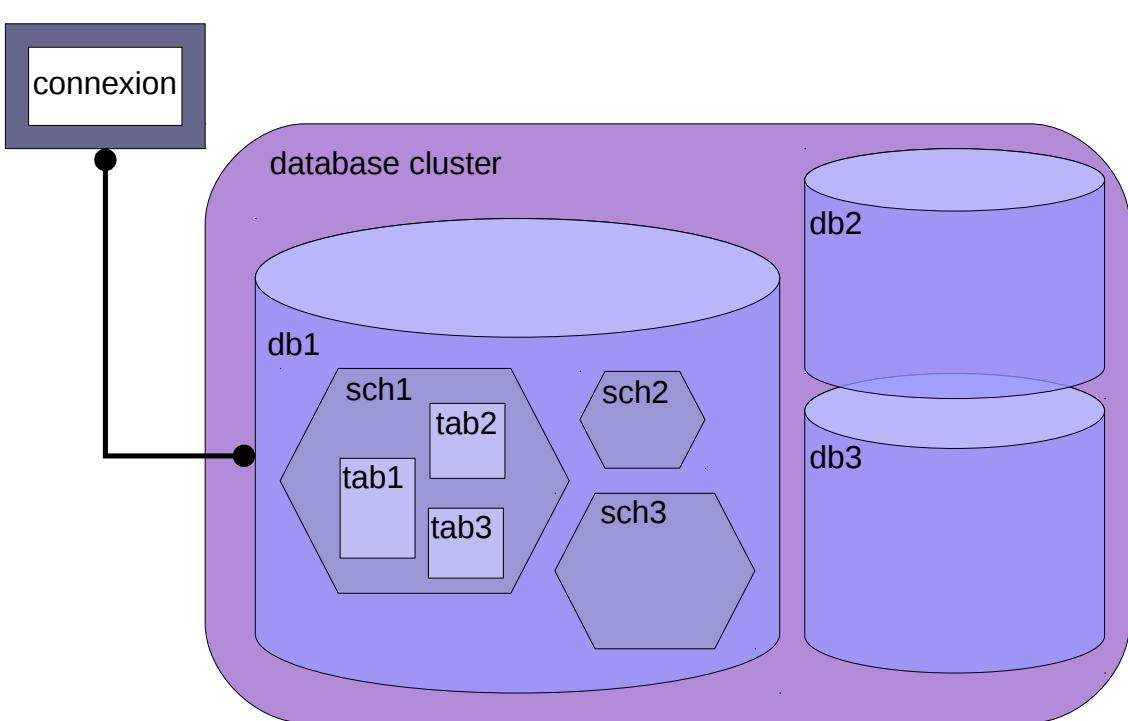


A PostgreSQL database cluster contains one or more named databases. Users and groups of users are shared across the entire cluster, but no other data is shared across databases. Any given client connection to the server can access only the data in a single database, the one specified in the connection request.

A database contains one or more named schemas, which in turn contain tables. Schemas also contain other kinds of named objects, including data types, functions, and operators. The same object name can be used in different schemas without conflict; for example, both schema1 and myschema can contain tables named mytable. Unlike databases, schemas are not rigidly separated: a user can access objects in any of the schemas in the database he is connected to, if he has privileges to do so.

<http://www.postgresql.org/docs/8.4/static/ddl-schemas.html>





Graphique 38 Organisation en cluster, base, schéma, table dans PostgreSQL

Syntaxe : Créer un schéma

```
CREATE SCHEMA myschema;
```

Syntaxe : Créer une table dans un schéma

```
CREATE TABLE myschema.mytable (
  ...
);
```

Syntaxe : Requêter dans un schéma

```
SELECT ...
FROM myschema.mytable
```



Exemple

Propriété	Valeur
Nom	test
OID	58807
Propriétaire	stc
ACL	
Droits par défaut pour les tables	
Droits par défaut pour les séquences	
Droits par défaut pour les fonctions	
Schéma système ?	Non
Commentaires	

```
-- Schema: test
-- DROP SCHEMA test;
CREATE SCHEMA test
    AUTHORIZATION stc;
```

Schéma sous PostgreSQL



Complément: Schéma par défaut

Afin d'alléger la syntaxe il est possible de définir un schéma par défaut, dans lequel seront créer les tables non-préfixées et un ou plusieurs schémas par défaut dans lesquels seront requêtées les tables non-préfixées.

```
SET search_path TO myschema,public;
```

Cette instruction définit le schéma myschema comme schéma par défaut pour la création de table et le requêtage, puis public pour le requêtage, le premier étant prioritaire sur le second :

- CREATE mytable créera ma mytable dans le schéma myschema.
- SELECT FROM mytable cherchera la table dans la schéma myschema, puis dans le schéma public si la table n'existe pas dans le premier schéma.



Remarque : Schéma "public"

Le schéma public est un schéma créé par défaut à l'initialisation de la base, et qui sert de schéma par défaut en l'absence de toute autre spécification.



Complément: Pour aller plus loin

<http://www.postgresql.org/docs/8.4/static/ddl-schemas.html>

6.5 PHP et BD

Objectifs

Comprendre le rôle des langages applicatifs (PHP par exemple) comme sur-couche au dessus des SGBD et SQL

Savoir accéder à une base de données depuis un programme PHP

6.5.1 Interfaçage avec PostgreSQL

 Syntaxe : Connexion à la BD

```
$vConn = pg_connect("host=$vHost port=$vPort dbname=$vDbname user=$vUser  
password=$vPassword");
```

 Syntaxe : Interrogation de la BD

```
$vSql ="SELECT ...";  
$vQuery=pg_query($vConn, $vSql);  
while ($vResult = pg_fetch_array($vQuery, null, PGSQL_ASSOC)) {  
    ... $vResult[nom_attribut]...  
}
```

 Syntaxe : Alimentation de la BD

```
$vSql="INSERT ...";  
$vQuery=pg_query($vConn, $vSql);
```

 Syntaxe : Déconnexion de la base de données

```
pg_close($conn)
```

6.5.2 Interfaçage PHP avec MySQL

Fait à partir de www.commentcamarche.net. Copyright 2003 Jean-François Pillou. Document soumis à la licence GNU FDL.
Complété à partir de MySQL 4 : Installation, mise en oeuvre et programmation [Thibaud03].

Connexion au serveur

```
mysql_connect($host,$user,$passwd);
```

Connexion à la base de données

```
mysql_select_db($bdd);
```

Exécution de requête SQL

```
$result=mysql_query($query)
```

Traitement de résultat de requête SELECT

```
/* Test d'exécution de la requête */  
if (! mysql_fetch_row($result)) {  
    echo "Aucun enregistrement ne correspond\n";  
}  
else {  
    while($row = mysql_fetch_row($result)) {  
        ... $row[1] ... $row[2] ...  
    }  
}
```

Déconnexion de la base de données

```
mysql_close();
```

6.5.3 Interfaçage PHP avec Oracle

Fait à partir de <http://www.php.net/manual/en/ref.oci8.php>.

 Remarque

L'API OCI a remplacé l'ancienne API ORA (qui n'est plus supportée dans PHP).



Syntaxe : Connexion à la base de données

oci_connect établit une connexion entre le serveur PHP et un serveur Oracle.

```
connection_id oci_connect(string username, string password, string bd);
```



Exemple

```
if (! $conn=oci_connect($user, $passwd, $bd)) {  
    echo "Impossible d'établir la connexion";  
    exit;  
}
```



Syntaxe : Préparation d'une requête

oci_parse analyse une requête SQL et retourne un pointeur sur un *statement* (espace de requête).

```
statement_handle oci_parse(connection_id connection, string query);
```



Exemple

```
if(! $statement=oci_parse($conn, $sql)) {  
    echo "Impossible de préparer la requête";  
    exit;  
}
```



Syntaxe : Exécution d'une requête

oci_execute exécute une commande déjà préparée avec OCI Parse. Il est possible de spécifier le mode d'exécution des transactions (par défaut, il est en auto-commit, c'est à dire que l'ordre commit est passé automatiquement après chaque instruction SQL). Il est préférable d'utiliser le mode OCI_DEFAULT qui permet de contrôler les commits.

```
boolean oci_execute(statement_handle statement, int mode);
```

\$mode permet de paramétriser le commit (par défaut, le commit est envoyé automatiquement si l'exécution est correcte).



Exemple

```
if(! oci_execute($statement, OCI_DEFAULT)) {  
    echo "Impossible d'exécuter la requête";  
    exit;  
}
```



Syntaxe : Commit d'une transaction

oci_commit valide la transaction en cours sur une connexion.

```
boolean oci_commit(connection_id connection);
```



Exemple

```
if(! oci_commit($conn)) {  
    echo "Impossible de valider la transaction";  
    exit;  
}
```



Remarque : Rollback

oci_rollback permet d'annuler une transaction.



Syntaxe : Récupération d'enregistrements

oci_fetch_array retourne la ligne suivante (pour une instruction SELECT) dans un tableau à une dimension (il écrasera le contenu du tableau s'il existe). Par défaut, le tableau sera un tableau à double index, numérique et associatif.

```
array oci_fetch_array(statement_handle statement)
```



Exemple

```
while ($row=oci_fetch_array($statement)) {
    echo $results[0];
    echo $results[1];
    ...
}
```



Syntaxe : Déconnexion de la base de données

oci_close ferme une connexion Oracle.

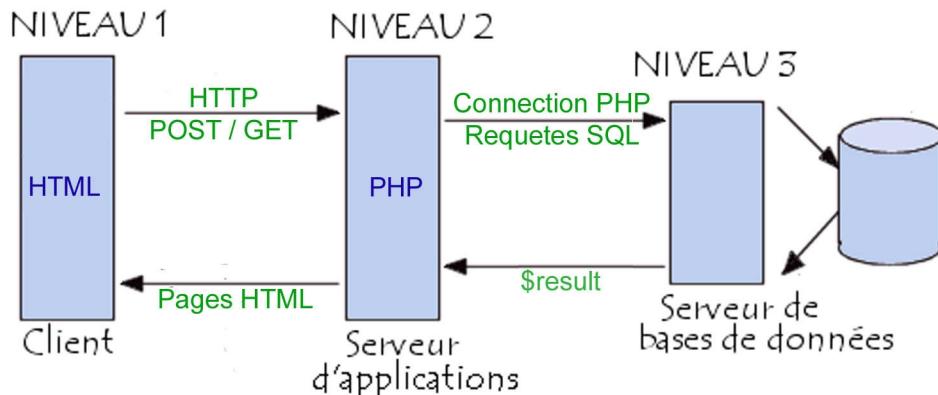
```
boolean oci_close(connection_id connection);
```



Exemple

```
if (! oci_close($conn)) {
    echo "Impossible de fermer la connexion ";
    exit;
}
```

6.5.4 Architecture PHP/Oracle



Navigateur Web

Apache + PHP

Oracle

Exemple d'architecture 3-tiers : PHP/Oracle (commentcamarche.net - © 2003 Pillou - GNU FDL)

6.6 Rappels Linux

6.6.1 Rappels architecture UTC

- Z:

6.6.2 Rappels Unix/Linux

- putty
- ssh
- commandes de base : cd, ~, man, ls, pwd, chmod, more

6.7 Bases de données et applications

6.7.1 Architecture générale d'une application de base de données



Définition

Une application de base de données comporte :

1. Une base de données : Elle pose le modèle de données, stocke les données, définit des vues sur les données (préparation des modalités de lecture)
2. Une application : Elle définit les interfaces homme-machine pour écrire et lire les données et contient le programme informatique de traitement des données avant insertion dans la base ou présentation à l'utilisateur.



Méthode : Procédure générale de développement

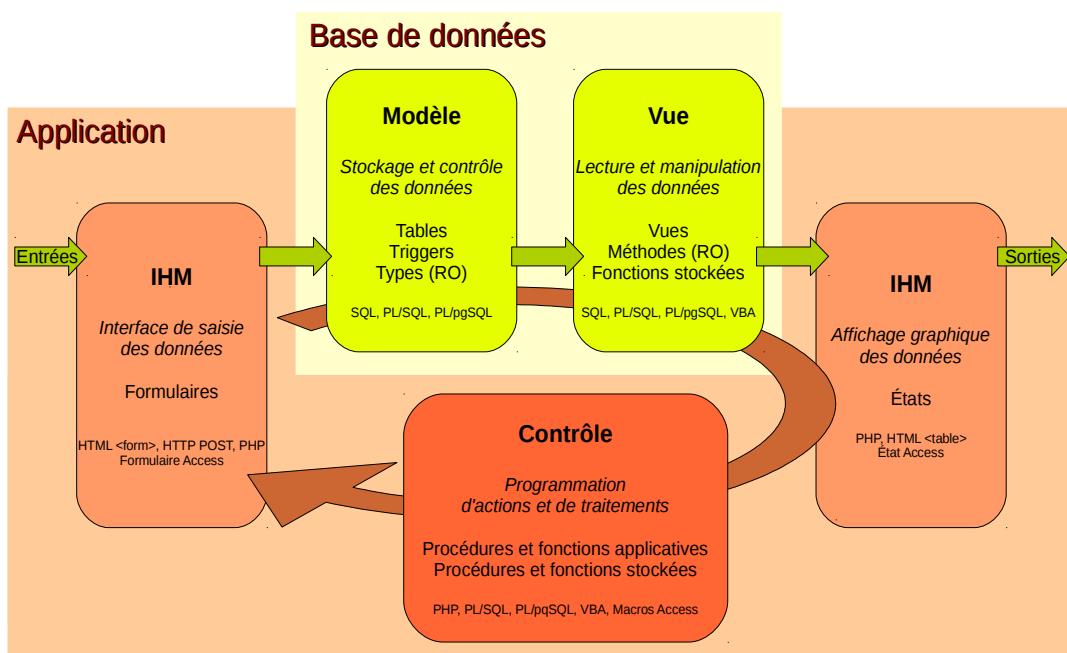
1. Développement de la BD : Conception en UML ou EA, puis traduction en R ou RO, puis implémentation SQL
 - Déclarer préalablement les types en RO
 - Ajouter les vues utiles, pour simplifier l'accès aux données en lecture
 - Ajouter des triggers, pour implémenter les contraintes complexes non exprimables en SQL (lorsque le SGBD le permet, comme Oracle ou PostgreSQL)
 - Implémenter les méthodes d'accès aux données (en RO), ou à défaut les fonctions stockées (en R) permettant de renvoyer les valeurs calculées prévues par le modèle conceptuel
2. Développement de l'application : traitements, formulaires et états
 - Les traitements proches des données sont réalisés dans le langage associé à la BD s'il existe (PL/SQL, PL/pgSQL, VBA)
 - Les traitements proches de l'application sont réalisés dans le langage applicatif choisi (PHP, VBA & Macros sous Access)



Exemple : Exemple de technologies

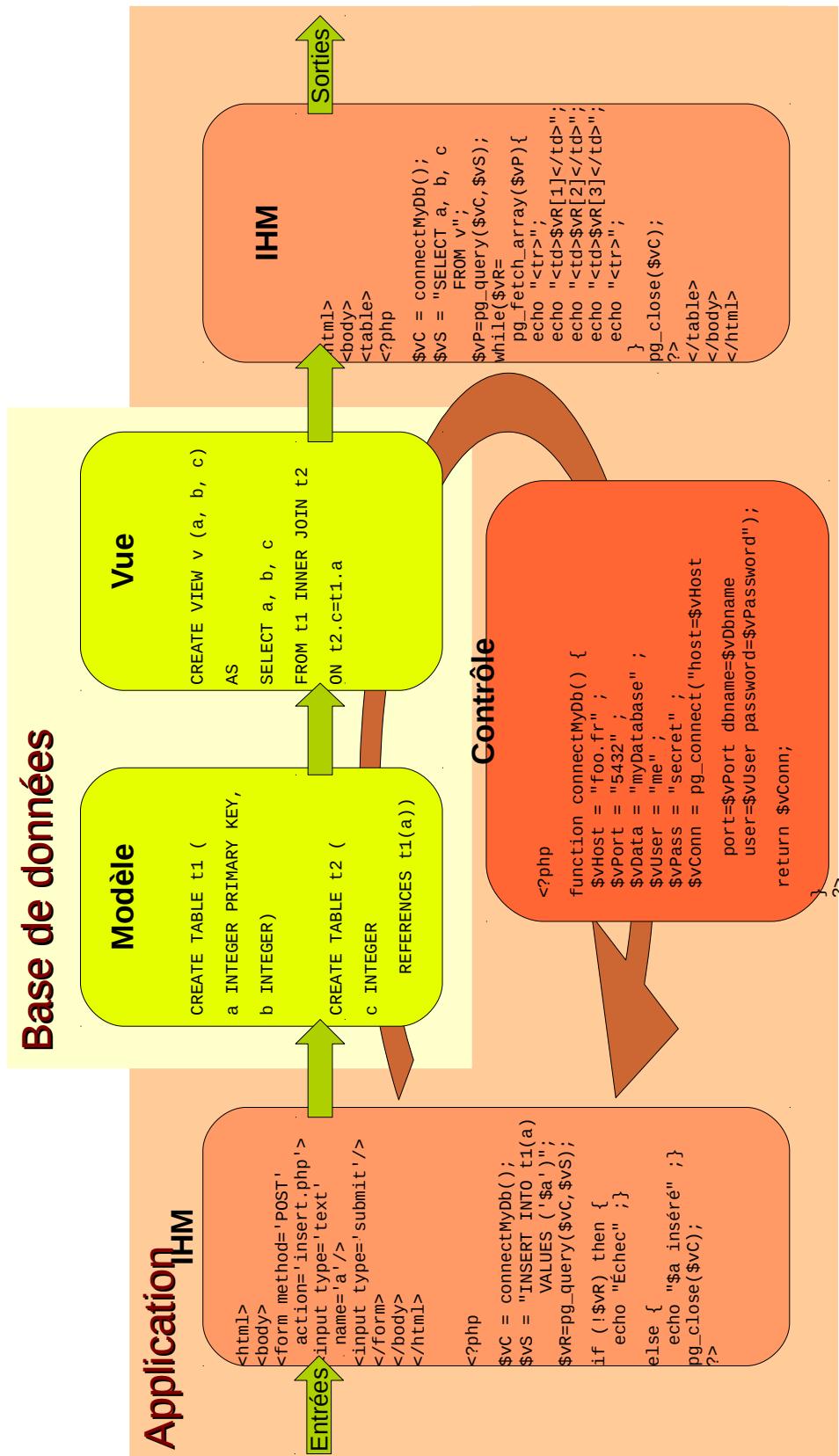
- BD :
 - SQL pour les tables, les types et les vues
 - PL/SQL (Oracle), PL/pgSQL (PostgreSQL) pour les triggers
 - PL/SQL, PL/pgSQL, VBA (Access) pour les fonctions stockées
- Application
 - HTML avec la balise `<form>` et le protocole HTTP/POST (Web), formulaires Access
 - PHP (echo) et présentation HTML (`<table>` par exemple), état Access
 - PL/SQL, PL/pgSQL, VBA (Access) pour les procédures et fonctions stockées (traitements proches de la base de données)
 - PHP (Web), Macros & VBA (Access)

Résumé en images



Graphique 39 Architecture générale d'une application de base de données

 Exemple : Exemple PHP/PosgreSQL



Graphique 40 Exemple d'application de base de données

6.7.2 Méthode générale d'accès à une BD en écriture par un langage de programmation



Méthode : Pseudo-code

1. Connexion à la base de données et récupération d'un identifiant de connexion
2. Écriture de la requête d'insertion ou de mise à jour de données
3. Exécution de la requête sur la connexion ouverte et récupération d'un résultat d'exécution (ici TRUE ou FALSE selon que la requête s'est exécuté ou non avec succès)
4. Test du résultat et dialogue avec l'utilisateur ou remontée d'erreur en cas de résultat FALSE
5. Clôture de la connexion

```
// Connexion à la base de données
$vHost = "foo.fr"
$vPort = "6666"
$vData = "myDatabase"
$vUser = "me"
$vPass = "secret"
$vConn = CONNECT ($vHost, $vPort, $vDb, $vUser, $vPass)
```

```
// Écriture de la requête
$vSql = "insert into t (a) values (1) ;"
```

```
// Exécution de la requête
$vResult = QUERY($vConn, $vSql)
```

```
// Test du résultat
IF (NOT $vResult) THEN MESSAGE("Échec de l'exécution")
```

```
// Clôture de la connexion
CLOSE ($vConn)
```

Remarque

 La connexion est bien entendu inutile dans le cas de procédure stockées, qui se trouvent par définition déjà associées à une BD en particulier.



Exemple : Fonction PHP

```
<?php
function fInsert ($pValue) {
// Connexion à la base de données
$vHost = "foo.fr" ;
$vPort = "5432" ;
$vData = "myDatabase" ;
$vUser = "me" ;
$vPass = "secret" ;
$vConn = pg_connect("host=$vHost port=$vPort dbname=$vDbname user=$vUser
password=$vPassword");
// Écriture, exécution et test de la requête
$vSql = "INSERT INTO t (a) VALUES ('$pValue')" ;
$vResult=pg_query($vConn, $vSql) ;
if (! $vResult) then {
    echo "Échec de l'insertion" ;
    return 0 ;
}
else {
    return 1 ;
}
// Clôture de la connexion
pg_close($vConn) ;
}
?>
```



Exemple : Procédure VBA

```
Sub fInsert(pValue As String)
    vSql = "INSERT INTO t (a) VALUES ('" & pValue & "')"
    CurrentDb.CreateQueryDef("", vSql).Execute
End Sub
```



Exemple : Programme Java

```
class fInsert {
    public static void main(String[] args) {
        try {
            // Connexion
            DriverManager.registerDriver ( new OracleDriver());
            Connection vCon =
DriverManager.getConnection("jdbc:oracle:thin:nf17/nf17@localhost:1521:test");
            // Exécution de la requête
            Statement vSt = vCon.createStatement();
            vSt.execute("INSERT INTO t (a) VALUES ('" + args[0] + "')");
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

6.7.3 Méthode générale d'accès à une BD en lecture par un langage de programmation



Définition : Pointeur sur un résultat de requête

Lorsqu'un programme exécute une requête dans une BD, il récupère un pointeur sur un résultat de requête permettant de parcourir le tableau résultant de l'exécution de la requête ligne par ligne.

Fn effet :

- Une requête SQL retourne toujours un tableau (même si parfois il n'a qu'une ligne et une colonne, c'est le fondement du modèle relationnel)
 - En général il n'est pas souhaitable que ce tableau soit transmis directement sous la forme d'une structure en mémoire (array), car le résultat d'une requête peut être aussi volumineux que la BD complète stockée en mémoire secondaire
 - La solution générale est donc que la BD fournit le résultat ligne par ligne

Le pointeur permet de :

- Retourner la ligne pointée sous la forme d'un vecteur (tableau à une seule ligne)
 - Passer à la ligne suivante
 - Savoir si l'on a atteint la fin du tableau (ligne N)

SELECT A, B, C, D FROM T

Pointeur 

	A	B	C	D
ligne 1				
ligne 2				
...				
...				
...				
...				
...				
...				
...				
ligne N				

Graphique 41 Pointeur sur un résultat de requête



Méthode : Pseudo-code

1. Connexion à la base de données et récupération d'un identifiant de connexion
 2. Écriture de la requête de sélection
 3. Exécution de la requête sur la connexion ouverte et récupération d'un pointeur sur un résultat de requête
 4. Parcours du pointeur dans une boucle permettant rapporter (*fetch*) et traiter (afficher par exemple) le tableau ligne par ligne
 5. Clôture de la connexion

```

// Connexion à la base de données
$vHost = "foo.fr"
$vPort = "6666"
$vData = "myDatabase"
$vUser = "me"
$vPass = "secret"
$vConn = CONNECT ($vHost, $vPort, $vDb, $vUser, $vPass)

// Écriture de la requête
$vSql = "select a, b from t;"

// Exécution de la requête
$vPointeur = QUERY ($vConn, $vSql)

// Traitement du résultat
IF (NOT $vPointeur) THEN MESSAGE("Échec de l'exécution")
ELSE
    WHILE ($vPointeur)
        FETCH $vPointeur IN $vResult[]
        PRINT $vResult[1]
        PRINT $vResult[2]
        NEXT $vPointeur
    END WHILE
END IF

// Clôture de la connexion
CLOSE ($vConn)

```

Exemple : Fonction PHP

```

<?php
function fInsert ($pValue) {
// Connexion à la base de données
    $vHost = "foo.fr" ;
    $vPort = "5432" ;
    $vData = "myDatabase" ;
    $vUser = "me" ;
    $vPass = "secret" ;
    $vConn = pg_connect("host=$vHost port=$vPort dbname=$vDbname user=$vUser
password=$vPassword");
// Écriture et exécution de la requête
    $vSql = "SELECT a, b FROM t;" ;
    $vPointeur=pg_query($vConn, $vSql) ;
// Affichage du résultat
    if (! $vPointeur) then {
        echo "Échec de la requête" ;
        return 0 ;
    }
    else {
        while ($vResult = pg_fetch_array($vQuery)) {
            echo "<p>$vResult[1] - $vResult[2]</p>";
        }
        return 1 ;
    }
// Clôture de la connexion
    pg_close($vConn)
}
?>

```

Exemple : Procédure VBA

```

Sub fSelect()
    vSql = "select a, b from t"
    Set vRs = CurrentDb.CreateQueryDef("", vSql).OpenRecordset
    Do While Not vRs.EOF
        Debug.Print vRs.Fields(0)
        Debug.Print vRs.Fields(1)
        vRs.MoveNext
    Loop
End Sub

```



Exemple : Programme Java

```
class fInsert {  
    public static void main(String[] args) {  
        try {  
            // Connexion  
            DriverManager.registerDriver (new OracleDriver());  
            Connection vCon =  
DriverManager.getConnection("jdbc:oracle:thin:nf17/nf17@localhost:1521:test");  
            // Exécution de la requête  
            Statement vSt = vCon.createStatement();  
            ResultSet vRs = vSt.executeQuery("SELECT a, b FROM t");  
            // Affichage du résultat  
            while(vRs.next()){  
                String vA = vRs.getString(1);  
                String vB = vRs.getString(2);  
                System.out.println(vA + " - " + vB));  
            }  
        }  
        catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

6.8 Bibliographie commentée sur les architectures Web et PHP



Complément: Références pratiques

PHP précis et concis [Lerdo01]

Une référence très rapide et pratique à utiliser sur PHP.

HTTP précis et concis [Wong00]

Petite référence pour comprendre rapidement les bases du protocole HTTP.

Apache précis et concis [Ford01]

Petite référence pour configurer un serveur Apache.

Access est un SGBDR du monde Microsoft Windows qui présente des particularités intéressantes dans le cadre de ce cours. Il propose une implémentation stricte des concepts relationnels, et ce dans un cadre facile d'accès technique : il constitue un bon moyen d'accès aux technologies des BD. C'est également un outil intéressant pour prototyper des **applications** de BD ou pour réaliser des applications finales dans des cadres d'usage restreint ou bureautique.

7.1 Introduction à Access

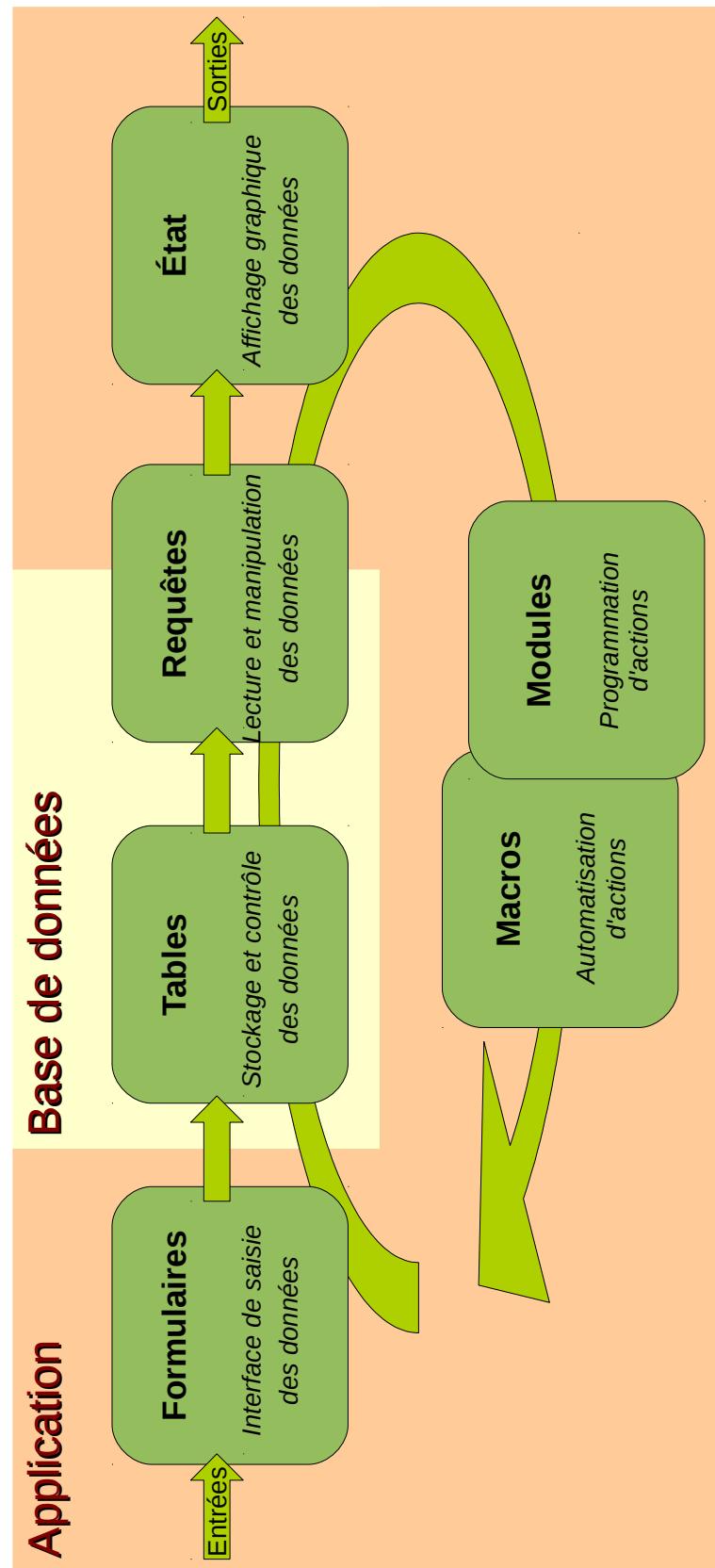
Objectifs

- Découvrir un SGBD simple d'usage
- Découvrir des principes de maquettage rapide d'application BD

7.1.1 Présentation d'Access

Access est un SGBDR et un outil de création d'application qui permet de :

- Créer des schémas relationnels et donc créer des **tables**, des contraintes sur les champs de ces tables et des contraintes référentielles entre ces tables
- Saisir des données dans les tables, avec l'instruction standard LMD INSERT ou à travers une interface graphique composée de **formulaires**
- Écrire des **requêtes** et des vues en utilisant le langage SQL ou bien le formalisme graphique QBE
- Réaliser des **formulaires** permettant d'alimenter ou interroger la BD
- Réaliser des **états** permettant de mettre en forme des résultats de requête de type SELECT
- Réaliser des **macros** permettant de programmer une application complète
- Réaliser des **modules** VBA permettant également de programmer une application complète, avec un spectre plus évolué que celui des macros



Graphique 42 Liste et fonctions des objets manipulables avec Access

7.1.2 Avantages et inconvénient d'Access

7.1.2.1 Avantages

- Rapidité de mise en oeuvre
- Facilité de maintenance ou reprise
- Rapidité de création d'IHM
- Langage graphique permettant un apprentissage rapide
- Schéma de données robustes (intégrité référentielle, contraintes, type de données, etc.)

7.1.2.2 Inconvénient

- Utilisation restreinte aux plate-formes Microsoft Windows
- Fiabilité douteuse
- Résistance faible à la montée en charge
- Peu adapté à des logiques réseaux
- Système de sécurité non standard, complexe et inadapté
- Faiblesse des IHM pour des applications complexes

7.1.2.3 Cas d'usage

Access est recommandé pour :

- L'apprentissage des BD
- Le prototypage rapide de BD et d'application (précision de cahier des charges, dialogue démonstratif avec les utilisateurs, phase avant la réalisation avec un SGBD industriel, etc.).
- Les petites applications locales ou LAN, avec peu d'utilisateurs (dizaines) et un volume de données raisonnable (centaines de milliers d'enregistrements, méga-octets).
- Les applications ne pouvant être maintenues par des informaticiens.

7.1.3 Séparation base de données et application

Conseil

 Access est à la fois un SGBDR® permettant de créer des BD® et à la fois un outil de développement d'application. Il est recommandé, pour des raisons méthodologiques et pratiques de bien séparer ces deux aspects du problème.

Méthode : Séparation BD / Application

 Pour créer une application complète sous Access, créer deux fichiers ".mdb", l'un contiendra la base de données (uniquement les tables et les vues sous forme de requêtes), l'autre l'application (les formulaires, états, macros et modules VBA).

Pour relier les deux applications, créer des tables liés dans le fichier "application", à l'aide du menu "Fichier / Données externes / Lier les tables".

Avantages de cette séparation

- **Séparation des problématiques de développement**
On ne fait pas à la fois le travail de modélisation de la BD et le travail de réalisation d'une application d'exploitation de cette BD.
- **Utilisation en réseau LAN**
Une BD centrale sur un serveur et N applications clientes locales.
- **Diminution des risques de crash**
Le crash de l'application cliente n'affecte pas la BD contenant les données.
- **Facilité de maintenance**
La mise à jour de l'application ne remet pas en cause la BD et ne nécessite pas de couper temporairement l'accès aux données. Les développements d'évolution de l'application Version N peuvent se poursuivre en parallèle de l'exploitation de la version N-1, sans avoir besoin de remettre à jour les données. L'extension du schéma relationnel peut se faire de façon transparente pour les applications.
- **Sécurité**
Plusieurs applications différentes peuvent utiliser la même base de données, tout en travaillant sur des tables différentes.

Remarque : Maintenance

 Toutes les modifications concernant la structure de la base de données (donc les tables) doivent être faites dans le fichier de la base de données et toutes les autres modifications doivent être faites dans le fichier de l'application (formulaires, états, etc.).

7.2 Crédation de schéma relationnel sous Access

Objectifs

Connaître les éléments techniques de base pour apprendre à créer une BD sous Access

7.2.1 LDD et création de tables sous Access

En général la création de la base se fera en utilisant l'interface graphique d'édition de table (mode **Création**), puis en déclarant les attributs un par un en mode interactif.



Remarque : Crédation du schéma en mode interactif

L'inconvénient de cette façon de créer la base est qu'il est impossible de recréer automatiquement le schéma de la base une seconde fois.



Remarque : Crédation du schéma en utilisant le LDD

Il est possible de créer le schéma relationnel dans Access en utilisant des instructions LDD. Il faut pour cela créer une nouvelle requête, passer en mode SQL et écrire chaque requête SQL une par une. Cette solution est donc peu efficace, à moins d'avoir écrit un petit programme VBA qui lit une suite d'instructions LDD et peut ainsi procéder à la constitution complète de la base à partir d'un seul fichier externe.

7.2.2 Domaines et types de données sous Access

Types de données

- Texte
- Numérique
- Date/Heure
- Oui/Non
- Monétaire
- NuméroAuto
- Mémo
- Objet OLE
- Lien hypertexte



Remarque : Domaine

Le domaine est un type de données pour lequel on a éventuellement ajouté certaines contraintes supplémentaires (telles que la taille du champs, la précision d'un numérique, des contraintes de validité restreignant les valeurs possibles, la présence ou non de la valeur de nullité, etc.)



Remarque : Énumération

Pour créer un type énumération, il faut partir d'un type de données standard, et restreindre le domaine aux valeurs autorisées spécifiées par l'énumération. On utilise pour cela la propriété "Valide si" relative à l'attribut concerné, avec une expression du type "Valeur1 ou Valeur2 ou ValeurN".

C'est équivalent à la clause CHECK en SQL standard.

7.2.3 Contraintes

Contraintes de tables et de colonnes

Access permet d'exprimer les contraintes SQL LDD standard :

- PRIMARY KEY : Sélectionner le ou les attributs concernés puis cliquer sur l'icône de clé primaire
- NOT NULL : Null interdit = oui dans les propriétés du champs
- Unique : Indexé = Oui - Sans doublons dans les propriétés du champs ou utiliser la fenêtre Index pour les clés composées de plusieurs attributs
- CHECK : Valide si dans les propriétés du champs ou de la table (pour une contrainte de table)



Attention: Enumération

Pour déclarer une énumération dans Access, il faut impérativement la spécifier au niveau de la structure de la base de données, donc au niveau de la propriété Valide si de l'attribut. La déclaration de cette énumération au niveau d'une liste déroulante dans l'IHM de saisie de la table (Liste de choix) n'est pas pertinente car c'est uniquement ce "petit bout d'IHM" qui contrôle les

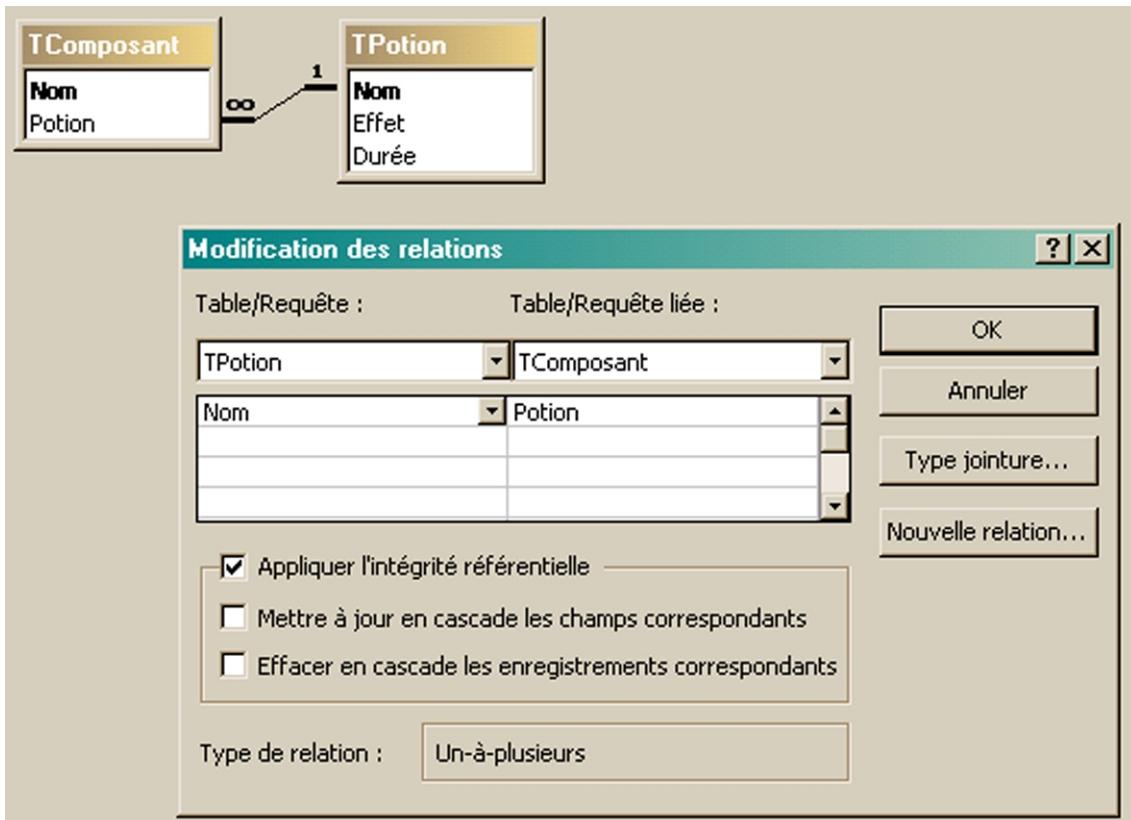
valeurs de l'enumération et non le moteur du SGBD. Donc une autre partie de l'application avec une autre IHM pourra permettre des valeurs différentes, ce qui n'est pas souhaité.

Contraintes d'intégrité référentielle

Access permet de spécifier les contraintes d'intégrité référentielle (clause FOREIGN KEY) à travers la fenêtre Relations (menu Outils de bases de données).

Pour spécifier une contrainte d'intégrité référentielle, il faut

1. afficher les tables concernées
2. glisser-déposer la clé étrangère sur la clé primaire
3. cocher la case Appliquer l'intégrité référentielle



Intégrité référentielle



Attention

Dans Access le mot "relation" désigne en fait les contraintes d'intégrités référentielles, et non les tables comme c'est le cas dans la terminologie relationnelle. Ceci est un abus de langage.



Rappel : Contrôle sur les données

Le contrôle sur les données opéré par un SGBD correspond à divers éléments : contrôle du domaine, mais aussi de l'intégrité référentielle, de la non nullité, etc.

Ces contraintes peuvent être levées : en spécifiant un domaine très permissif (comme les chaînes de caractères), en supprimant les clause de non nullité, ou bien en supprimant l'intégrité référentielle (dans la fenêtre Relations d'Access).

La suppression des contraintes peut être nécessaire pour intégrer temporairement des données qui ne correspondent pas au modèle. Elles devront bien entendu dans ce cas être mises en conformité au plus vite, et les contraintes pourront alors être réactivées.

7.2.4 Vues et "requêtes" LDD

Il est possible de spécifier des créations de schémas relationnels en utilisant le LDD de SQL classique sous Access.

Pour cela il faut créer un objet Requête, puis indiquer que c'est une requête de type Création de table.



Attention: Types non reconnus

Access n'accepte pas toutes les déclarations de type dans ce cadre. Par exemple il n'est pas possible de déclarer INTEGER(X), mais seulement INTEGER. La contrainte de taille ne pouvant se faire qu'en mode interactif.

Vues

Sous Access, les objets Requêtes sont enregistrés.

Pour créer une vue, il suffit donc de créer un objet Requête de Sélection et de l'enregistrer.

En fait, sous Access, la norme et la vue et l'exception la requête au sens d'évanescence (pour exécuter une requête qui ne soit pas enregistrée comme une vue, il suffit de ne pas enregistrer la requête).

7.3 Le langage de requêtes sous Access

Objectifs

Connaître les principes du langage de requête QBE et savoir lire une requête QBE

7.3.1 Questions SQL

Pour écrire une requête en SQL sous Access, il suffit de créer objet requête, puis de passer en mode SQL (menu Créer).

Il est ainsi possible d'écrire du LDD (CREATE TABLE) et du LMD (SELECT, INSERT, UPDATE).

 Remarque : Équivalence QBE / SQL

Il est possible de faire des aller-retour entre le mode SQL et le mode Création (QBE).

7.3.2 Manipulation des données en QBE

Pour exécuter une requête autre que de type SELECT, il faut changer son type via le menu Créer lorsque la requête est en mode Création.

- **UPDATE** : Mise à jour
- **INSERT** : Ajout
- **DELETE** : Suppression



Changer le type de requête

 Attention: Problèmes de traduction

Des problèmes de traduction dans certaines versions d'Access conduisent à des formulations hasardeuses des menus. Ainsi "Mettre à jour une requête" signifie "Requête de mise à jour" et "Supprimer une requête" signifie "Requête de suppression". L'erreur de traduction vient probablement du double sens de "Update query" et "Delete query" en anglais...

Ces défaut sont supprimés dans les versions récentes d'Access.

7.3.3 Clause GROUP BY en QBE

Appliquer la clause GROUP BY se nomme en Access "faire un regroupement". L'interface QBE ne le permet pas par défaut, il faut au préalable afficher la ligne Opération, qui fera apparaître une ligne supplémentaire sur l'interface (menu Affichage > Opérations avant Access 2007, bouton Totaux depuis)



Activer le GROUP BY



Exemple : Regroupement en QBE sous Access et GROUP BY en SQL

Champ :	Effet	Durée
Table :	TPotion	TPotion
Opération :	Regroupement	Moyenne
Tri :		
Afficher :	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Critères :		

Regroupement en QBE

```
SELECT Effet, Avg(Durée) AS MoyenneDeDurée
FROM TPotion
GROUP BY Effet;
```



Remarque : HAVING

Pour ajouter une clause HAVING, il suffit d'utiliser la ligne "critère" sous une propriété à laquelle est appliquée une opération.

7.4 Création d'application Access (formulaires et macros)

Objectifs

Connaître les éléments techniques de base pour apprendre à créer une application sous Access

Notons que les états ne sont pas traités ici, mais leur apprentissage est aisément compris pour les formulaires.

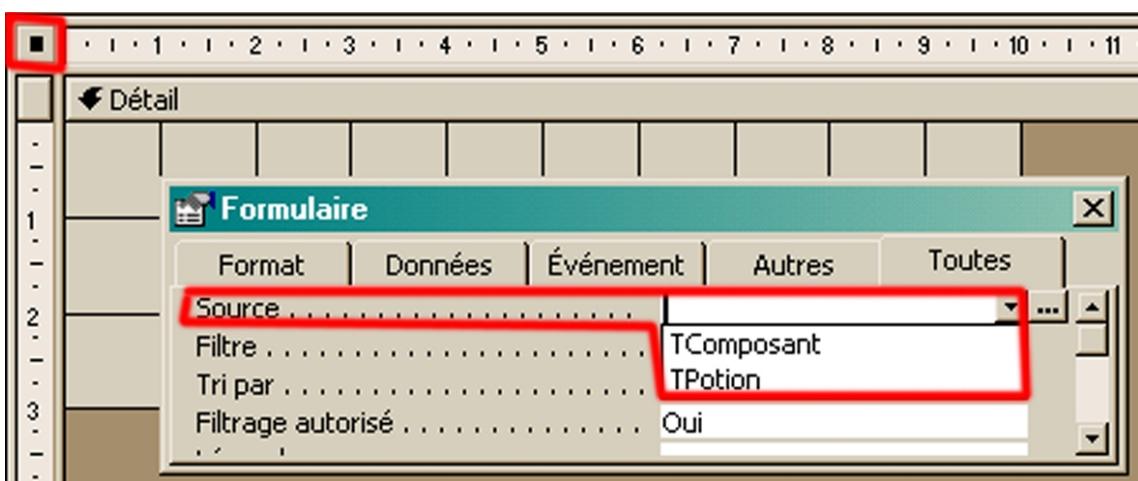
7.4.1 Formulaire liés à une table



Définition : Formulaire lié

Un formulaire lié est un formulaire qui offre un accès direct à une (et une seule) table.

Pour lier un formulaire à une table il faut désigner le nom d'une table dans la propriété "Source" du formulaire.

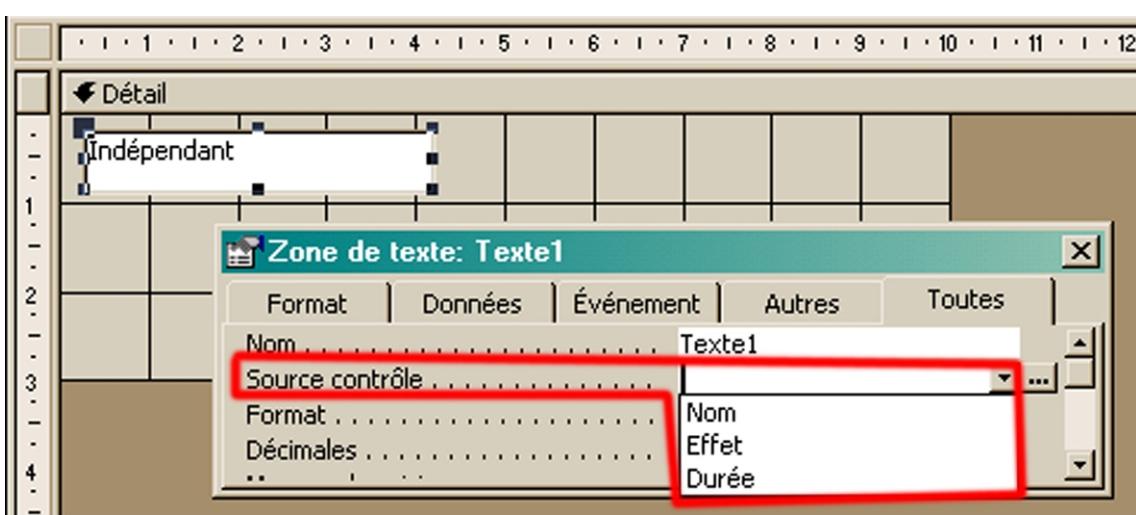


Source d'un formulaire



Remarque : Contrôle lié

Dans le cadre d'un formulaire lié, il est possible (et c'est même le seul intérêt) de créer des contrôles liés. De tels contrôles référencent directement un attribut d'une table (grâce à leur propriété "Source contrôle") et permettent donc une saisie directe d'information dans la base sans écrire de code SQL ou LMDI.



Source d'un contrôle



Remarque : Usage des formulaires liés

Il est plutôt déconseillé de recourir aux formulaires liés, qui, s'ils offrent une première approche très simple pour créer des interfaces de saisie, reste très limités fonctionnellement. On leur préférera rapidement les formulaires indépendants et les requêtes LMDI de type INSERT et UPDATE.



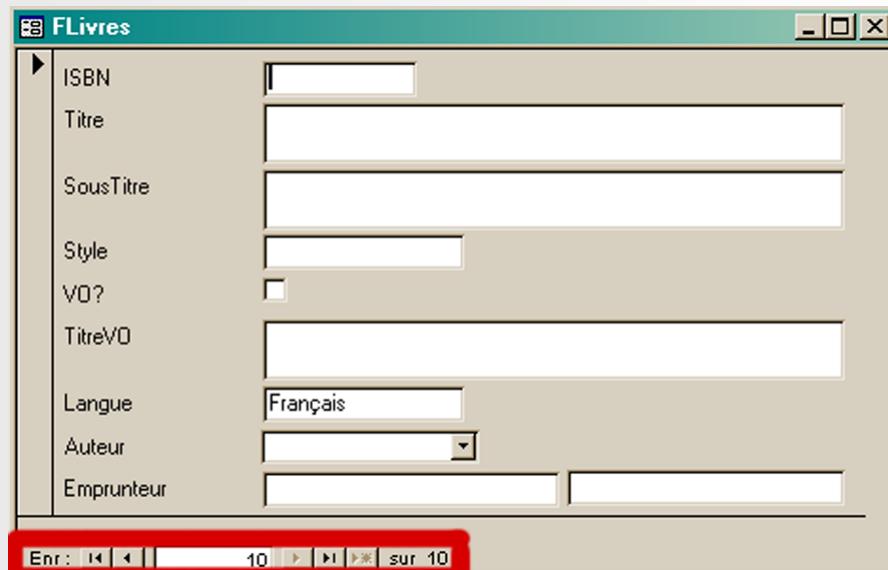
Attention: Formulaire lié et pages multiples

Un formulaire lié, lorsqu'il est en "Mode simple" (valeur de la propriété "Affich par défaut") comporte autant de "pages" qu'il y a d'enregistrements dans la table liée.

Chaque page permet de modifier l'enregistrement qu'elle matérialise, et la dernière page permet d'ajouter un nouvel enregistrement.

Par défaut le formulaire s'ouvre toujours sur le **premier** enregistrement, et il faut donc se déplacer après le dernier pour en ajouter un nouveau.

Il faut pour cela utiliser les boutons en bas à gauche du formulaire :



Déplacement dans les différents enregistrements d'un formulaire lié



Définition : Formulaire indépendants

Les formulaires indépendants ne sont pas liés à une table (leur propriété "Source" est vierge).

Ils servent à récupérer des valeurs en mémoire, dans des variables, avant d'en faire un traitement par programmation : On pourra par exemple utiliser cette valeur dans une requête SQL LMDI de type INSERT pour ajouter des enregistrements dans une table.

Il est toujours possible de faire, avec un formulaire indépendant et une requête SQL, ce qu'il est possible de faire avec un

formulaire lié (et l'on peut bien entendu faire beaucoup plus).



Remarque : Contrôles et formulaires indépendants

Les contrôles d'un formulaire indépendant ne peuvent bien entendu pas être liés à une table. Ils ne peuvent servir qu'à stocker une valeur en mémoire pour un usage donné. Ils sont de ce fait comparable à des **variables**.



Syntaxe : Utiliser une valeur saisie dans le contrôle d'un formulaire indépendant

Pour utiliser les valeurs indépendantes (en mémoire) de contrôle d'un formulaire, dans d'autres objets (formulaires, états, requêtes, macros ou modules), utiliser l'une des deux syntaxes suivantes :

```
Formulaires!NomDuFormulaire!NomDuContrôle
```

```
Forms!NomDuFormulaire!NomDuContrôle
```



Exemple : Mise à jour de plusieurs tables

1. Saisie de valeurs relatives à plusieurs enregistrements, de tables différentes, dans un formulaire
2. Utilisation de ces valeurs dans des requêtes d'insertion pour ajouter les valeurs dans les tables

```
INSERT INTO TVilles (pkVille, aPays)  
VALUES (Formulaires!FChoix!Ville, Formulaires!FChoix!Pays)
```



Exemple : Requêtes paramétrées

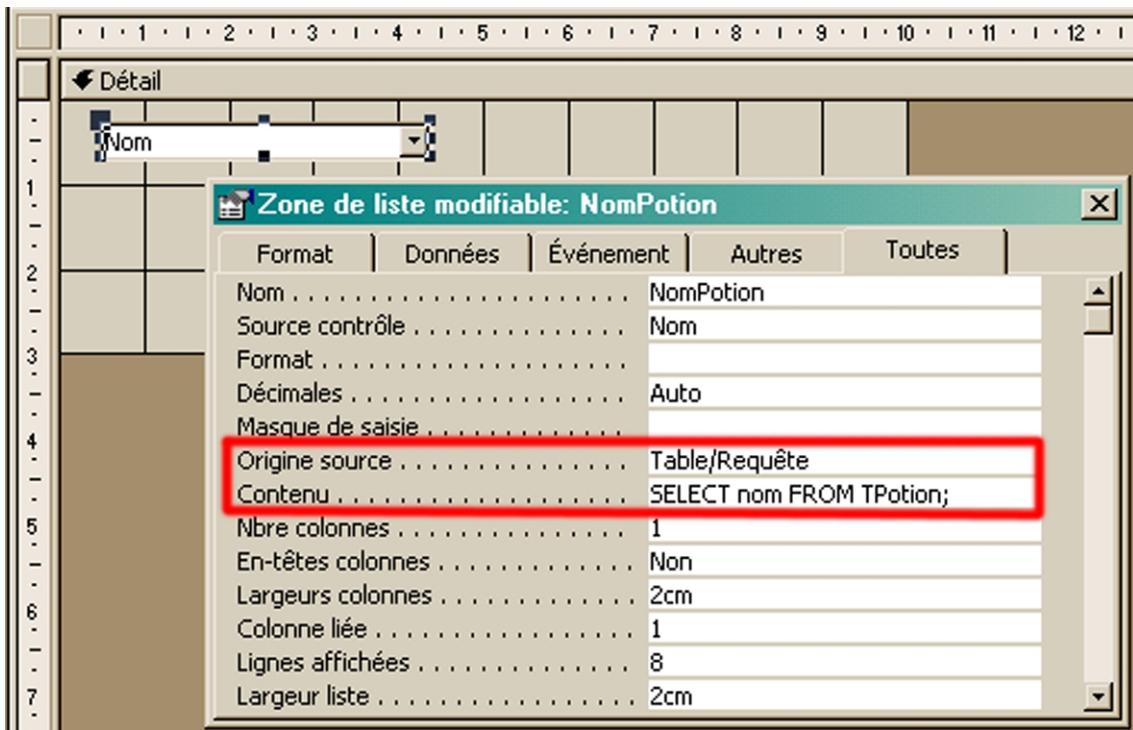
1. Saisie de valeurs relatives à des paramètres dans un formulaire
2. Exécution d'une requête de sélection s'appuyant sur ces paramètres

```
SELECT * FROM TVilles  
WHERE Pays = Formulaires!FChoix!Pays
```

7.4.3 Contrôles listes

Les listes déroulantes sont des contrôles très intéressants qui permettent de récupérer des valeurs dynamiques grâce à une requête ou de spécifier des valeurs statiques parmi lesquelles l'utilisateur choisira.

- Pour que les valeurs de la liste soit définies dynamiquement à partir du résultat d'une requête, fixer la propriété "Origine source" à la valeur "Table/Requête", puis écrire la requête dans la propriété "Contenu".
- Pour que les valeurs de la liste soit définies statiquement, fixer la propriété "Origine source" à la valeur "Liste Valeurs", puis lister les valeurs en les séparant par des points-virgules dans la propriété "Contenu".



Contenu d'un contrôle liste

 Remarque : Liste à plusieurs colonnes

Il est possible de réaliser des listes déroulantes à plusieurs colonnes. Il suffit pour cela que la requête spécifiée dans "Contenu" projette plusieurs attributs.

Il faudra dans ce cas régler correctement les propriétés "Nbre colonnes" (le nombre de colonnes) et "Colonne liée" (le numéro de la colonne principale, qui renvoie la valeur à stocker réellement dans la variable associée au contrôle).

 Remarque : Récupération de la valeur d'une colonne d'une liste

```
Formulaires!NomDuFormulaire!NomDuContrôleListe.column(X)
Avec X de 0 à N-1, N étant le nombre de colonnes
```

7.4.4 Macros

 Attention

Un objet "macro" est en fait un objet "groupe de macro". Il faudra afficher les colonnes optionnelles :

- Noms de macros
- Conditions
- Arguments

Une macro est définie par un nom (première colonne) et une succession d'instructions (actions dans la dernière colonne) éventuellement soumises à des conditions d'exécution (seconde colonne). Chaque action requiert de fixer un certain nombre de paramètres qui s'affiche dans la partie basse de l'interface après que l'action ait été sélectionnée.

Exemple



- **OuvrirFormulaire**
Ouvre un formulaire
 - **OuvrirEtat**
Ouvre un état
 - **BoîteMsg**
Crée une boîte de dialogue avec l'utilisateur.
 - **AtteindreEnregistrement**
Dans un formulaire lié, permet d'atteindre un enregistrement particulier.
 - **DéfinirValeur**
Permet de fixer la valeur de n'importe quelle propriété d'un contrôle de formulaire. Cette action est très utile pour rendre les interfaces plus dynamiques.
 - **ExécuterMacro**
Exécute une autre macro. Cette action est utile pour modulariser le code Macro (bien que sans passage de paramètres, cela reste sommaire).
 - **Fermer**
Ferme un objet de type formulaire, état, etc.
 - **TrouverEnregistrement**
Dans un formulaire lié, permet de se rendre à un enregistrement particulier en fonction de la valeur de l'un de ses contrôles.
 - **AtteindreContrôle**
Permet de sélectionner un contrôle particulier dans un formulaire. Cette action est utile avant d'effectuer un TrouverEnregistrement par exemple.
 - **Actualiser**
Permet de rafraîchir un contrôle (après un DéfinirValeur ou pour ré-exécuter la requête source d'une liste déroulante par exemple)
 - **Avertissements**
Active ou désactive les avertissements lors de l'exécution de requêtes.
 - **ExécuterCommande**
Permet d'exécuter une des commandes disponible dans les menu d'Access
 - **ArrêtMacro**
Stoppe la macro, sans exécuter les instructions restantes. Cette actions est utile pour terminer la macro après un test par exemple.

Appel des macros

Les macros sont en général appelées par des évènements particuliers survenus lors de la manipulation des formulaires et états (programmation évènementielle des formulaires et états).

Les macros peuvent également être appelées par d'autres macro, voire du code VBA.



Complément : Macro AutoExec

Si une macro porte le nom "AutoExec", elle sera exécutée automatiquement à l'ouverture de la base de données.

Cette macro peut servir typiquement à afficher un menu général d'entrée dans l'application.

Pour désactiver l'exécution automatique de la macro AutoExec, maintenez la touche "shift" de l'ordinateur appuyée lors du lancement de l'application.

7.5 Modules de programmation VBA sous Access

Objectifs

Découvrir les modalités d'accès à une base de données via un langage de programmation
Savoir accéder à une base de données Access depuis un programme VBA

7.5.1 Structure d'un programme VBA

- Fonction (function)**

Retourne une valeur, utilisable dans les requêtes, les formulaires et les états.

- Procédure (sub)**

Effectue des opérations, en général sur les tables, notion de procédures stockées

Syntaxe

```
{Sub | Function} nom (param1 As type, ...) {As type retourné}
Dim variable1 As type
...
programme
{End Sub | End Function}
```

7.5.2 Fonctions à connaître

- Traitement de chaîne : LCase(S), UCase(S), Left(S, 1), Right(S, 1), Len(S), +, ...
- Génération aléatoire : Randomize, Rnd
- Gestion nombre : Cast : Int(),

7.5.3 Objets VBA pour accéder à la BD

- CurrentDb : Objet base de données
 - .CreateQueryDef(nom_requête, code_sql) : Crée un objet requête "nom_requête" avec le code "code_sql". Si "nom_requête" est renseigné la requête est enregistrée comme un objet persistant dans la base, si "nom_requête" est vide la requête n'est pas enregistrée.
- QueryDef : Objet requête
 - .OpenRecordset : Exécute une requête de type SELECT
 - .Execute : Exécute une requête de type INSERT, UPDATE ou DELETE (ou LDD)
- Recordset : Pointeur sur un résultat de requête
 - .MoveNext, .MoveFirst, .MoveLast : Atteindre l'enregistrement suivant, le premier, le dernier
 - .RecordCount : Le nombre d'enregistrements du résultat
 - .EOF : Booléen permettant de tester si on a atteint à la fin du résultat
 - !nom_champ : La valeur du champ "nom_champ"

Syntaxe : Alimentation de la BD

```
CurrentDb.CreateQueryDef("", "INSERT...").Execute
```

Syntaxe : Interrogation de la BD

```
Set vRs = CurrentDb.CreateQueryDef("", "SELECT...").OpenRecordset
Do While Not vRs.EOF
  ... vRs!MonChamp ...
  vRs.MoveNext
Loop
```

Remarque : Approche alternative

```
For i = 1 To vRs.RecordCount
  ... vRs!MonChamp ...
```

```
vRs.MoveNext
Next i
```

7.5.4 Exemple : Normaliser des chaînes de caractères

```
Function NormaliserChaine(s As String) As String
    Result = ""
    For i = 1 To Len(s)
        C = Right(Left(s, i), 1)
        Select Case C
            Case "é", "è", "ê"
                C = "e"
            Case "à"
                C = "a"
            Case "í"
                C = "i"
        End Select
        Result = Result + C
    Next i
    Result = UCase(Left(Result, 1)) + LCase(Right(Result, Len(Result) - 1))
    NormaliserChaine = Result
End Function
```

7.5.5 Exemple : Accéder à un fichier externe

Cet exemple (extrait de l'aide en ligne d'Access) utilise la fonction Input pour lire un fichier caractère par caractère et les afficher dans la fenêtre Exécution. Nous supposons que FICHTEST est un fichier texte comportant quelques lignes de données.

```
Sub LireFichier()
    Dim MyChar
    Open "FICHTEST" For Input As #1 ' Ouvre le fichier
    Do While Not EOF(1) ' Effectue la boucle jusqu'à la fin du fichier
        MyChar = Input(1, #1) ' Lit un caractère
        Debug.Print MyChar ' Affiche dans la fenêtre Exécution
    Loop
    Close #1 ' Ferme le fichier
End Sub
```

7.5.6 Exemple : Parcourir une table ou une requête stockée

```
Sub ParcoursTable()
    Set vRs = CurrentDb.OpenRecordset("MaTable")
    For i = 1 To vRs.RecordCount
        Debug.Print vRs!MonChamp
        vRs.MoveNext
    Next i
End Sub
```

7.5.7 Exemple : Parcourir une table passée en paramètre

```
Sub ParcoursTable(pTable As String, pChamp As String)
    Set vRs = CurrentDb.OpenRecordset(pTable)
    Set vChamp = vRs.Fields
    For i = 0 To vChamp.Count - 1
        If vChamp(i).Name = pChamp Then NumChamp = i
    Next i
    For i = 1 To vRs.RecordCount
        Debug.Print vRs.Fields(NumChamp)
        vRs.MoveNext
    Next i
End Sub
```

7.5.8 Exemple : Parcourir une table et gérer les erreurs

```
Sub ParcoursTable(pTable As String, pChamp As String)
On Error GoTo Erreur_ParcoursTable
Set vRs = CurrentDb.OpenRecordset(pTable)
Set vChamp = vRs.Fields
For i = 0 To vChamp.Count - 1
    If vChamp(i).Name = pChamp Then NumChamp = i
Next i
For i = 1 To vRs.RecordCount
    Debug.Print vRs.Fields(NumChamp)
    vRs.MoveNext
Next i
Exit_ParcoursTable:
Exit Sub
Erreur_ParcoursTable:
Debug.Print "Erreur"
MsgBox "Erreur"
End Sub
```

7.5.9 Exemple : Exécuter une requête

```
Sub ExecuterRequete(pTable As String, pChamps As String)
Dim vCodeSql As String

vCodeSql = "select " + pChamps + " from " + pTable
Set vRs = CurrentDb.CreateQueryDef("", vCodeSql).OpenRecordset
For i = 1 To vRs.RecordCount
    Debug.Print vRs.Fields(0)
    vRs.MoveNext
Next i
End Sub
```

7.5.10 Exemple : Créer un schéma de BD et initialiser les données

```
Sub InitBD()
Const NbLignesSql As Integer = 3
Dim vCodeSql(NbLignesSql) As String
vCodeSql(1) = "drop table Table1"
vCodeSql(2) = "create table Table1 (Champs1 String(150))"
vCodeSql(3) = "insert into Table1 (Champs1) values ('test')"
For i = 1 To NbLignesSql
    CurrentDb.CreateQueryDef("", vCodeSql(i)).Execute
Next i
End Sub
```

7.5.11 Aide et déboguage



Attention: Accéder à l'aide en ligne

Pour accéder à l'aide en ligne sur les fonctions VBA demander l'aide (F1) depuis l'éditeur Visual Basic (Module) et non directement depuis Access.



Attention: Fenêtre d'exécution

La fenêtre d'exécution est indispensable pour tester les programmes VBA.

Pour obtenir la fenêtre d'exécution, faire "Affichage / Fenêtre d'exécution" dans le menu de l'éditeur VBA / Module.

La syntaxe "Debug.Print" dans un programme permet ensuite d'afficher des informations dans cette fenêtre (valeur de variable typiquement).

7.6 Autres aspects

7.6.1 Gestion des droits

Gestion des droits

- Non standard
- Fichier .MDW
- Pas simple ...

7.6.2 Run-time

- Ca existe

7.7 En résumé : Access

Access

À la fois un SGBDR pour créer des BD et un outil de développement d'applications

- BD
 - Dimension relevant de la conception de BD
 - LDD
 - Tables
 - LMD
 - Requêtes (Vue)
 - Application
 - Dimension relevant de la conception d'application.
 - IHM
 - Formulaires
 - États
 - Programmation
 - Macros
 - Modules VBA

7.8 Bibliographie commentée sur Access



Complément: Pratique

Comprendre les jointures dans Access [w_mhubiche.developpez.com]

Un tutoriel très pédagogique sur l'expression de jointures sous Access, qui peut par ailleurs servir à comprendre l'opération de jointure en général si besoin.



Complément: Pour aller plus loin

Developpez.com [w_access.developpez.com]

Un portail avec de nombreuses ressources.

FAQ de Développez.com [w_access.developpez.com/faq]

Une base de questions très intéressante pour la dimension outils de création d'application d'Access. On consultera en particulier les informations sur la gestion de la sécurité, les formulaires, VBA.

7.9 Questions-réponses sur Access

Comment supprimer les messages d'alertes lors de l'exécution des requêtes ?

VBA : DoCmd.SetWarnings False

Macro : Avertissements=Non

Comment atteindre un contrôle d'un sous-formulaire ?

Forms! [NomFormulaire].Form! [NomSousFormulaire]! [MaZoneDeTexte]

(ici pour une zone de texte)

Comment parcourir un "RecordSet" en VBA?

```
Rst.MoveFirst  
While not rst.EOF  
    ' code  
    rst.MoveNext  
Wend
```

```
Do Until rst.EOF  
    ' code  
    rst.MoveNext  
Loop
```

Comment calculer la difference entre 2 dates en VBA ?

```
DateDiff("d", date1, date2)  
'donne le nombre de jour entre date1 et date2
```

Comment mettre un point d'arrêt dans un code VBA ?

Quand Access passe sur un point d'arrêt l'exécution du code s'arrête et attends une manœuvre de votre part pour continuer.

- Quand le code est arrêté vous pouvez connaître la valeur des variables de votre code simplement en passant le pointeur de la souris dessus.
Vous pouvez choisir de continuer à exécuter le code en mode pas à pas en appuyant sur *F8*.
Vous pouvez poursuivre l'exécution du code normalement avec *F5*.

Quel est la différence entre "." et "!" ?

L'opérateur "/" (point d'exclamation) indique que l'élément qui suit est défini par l'utilisateur (un élément d'une collection). Par exemple, vous pouvez utiliser l'opérateur "/" pour faire référence à un formulaire ouvert, à un état ouvert, ou à un contrôle figurant sur un formulaire ou sur un état.

L'opérateur "." (point) indique généralement que l'élément qui suit est défini par Microsoft Access. Par exemple, vous pouvez utiliser l'opérateur "." pour faire référence à une propriété d'un formulaire, d'un état, ou d'un contrôle.

Glossaire

Client

Un client est un programme informatique qui a pour fonction d'envoyer des requêtes à un autre programme informatique, appelé serveur, d'attendre le résultat de cette requête et de traiter les résultats de la requête. Notons qu'un programme peut-être client vis à vis d'un programme et serveur vis à vis d'un autre. On ne prend pas ici le terme client dans son acception matérielle, qui signifie alors un ordinateur qui a pour fonction d'héberger des programmes clients.

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.

Extension

L'extension est l'explicitation d'un domaine par l'énonciation exhaustive de l'ensemble des objets du domaine.

Elle s'oppose à l'intension qui est une description abstraite des caractéristiques du domaine.

- Exemple : {bleu, rouge, vert}
- Contre-exemple : Le domaine des couleurs

Intension

L'intension est l'explicitation d'un domaine par la description de ses caractéristiques (en vue de sa compréhension abstraite, générale).

Elle s'oppose à l'extension qui est l'énonciation exhaustive de l'ensemble des objets du domaine.

- Exemple : Le domaine des couleurs
- Contre-exemple : {bleu, rouge, vert}

RAID

La technologie RAID permet de repartir 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 sûreté (en repartissant 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.

Sérialisation

Processus consistant à enregistrer des données en mémoire vive (par exemple des objets) sous une forme permettant leur persistance, typiquement sur une mémoire secondaire.

Serveur

Un serveur est un programme informatique qui a pour fonction de recevoir des requêtes d'un autre programme, appelé client, de traiter ces requêtes et de renvoyer en retour une réponse. Notons qu'un programme peut-être serveur vis à vis d'un programme et client vis à vis d'un autre. On ne prend pas ici le terme serveur dans son acception matérielle, qui signifie alors un ordinateur qui a pour fonction d'héberger des programmes serveurs.

XSL-FO

XSL-FO (FO pour Formatting Objects) est la seconde partie du standard W3C « Extensible Stylesheet Language Family ». Il propose un langage XML de mise en forme de documents imprimables.

Exemple d'extrait de code FO :

- <fo:block font-family="Times" font-size="12pt">Ceci est un paragraphe en police Times de taille 12pt</fo:block>.

Signification des abréviations

- 1NF	First Normal Form
- 2NF	Second Normal Form
- 3NF	Third Normal Form
- 4NF	Fourth Normal Form
- 5NF	Fifth Normal Form
- ANSI	American National Standards Institute
- BCNF	Boyce-Codd Normal Form
- BD	Base de Données
- CSV	Comma Separated Values
- DF	Dépendance Fonctionnelle
- DFE	Dépendance Fonctionnelle Élémentaire
- E-A	Entité-Association
- E-R	Entity-Relationship
- HTML	HyperText Markup Language
- IHM	Interaction Homme Machine ou Interface Homme Machine
- ISO	International Standardization Organization
- 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
- OS	Operating Système (Système d'Exploitation)
- PSM	Persistent Stored Modules
- QBE	Query By Example
- R	Relationnel
- RO	Relationnel-Objet
- 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
- SGML	Standard Generalized Markup Language
- SI	Système d'Information
- SQL	Structured Query Language
- UML	Unified Modeling Language
- W3C	World Wide Web Consortium
- XML	eXtensible Markup Language

Bibliographie

- [André89] JACQUES ANDRÉ, RICHARD FURUTA, VINCENT QUINT, *Structured documents*, Cambridge University Press, 1989.
- [Brillant07] ALEXANDRE BRILLANT, *XML : Cours et exercices*, Eyrolles, 2007 [ISBN 978-2212126914]
- [Celko00] CELKO JOE. *SQL avancé : Programmation et techniques avancées*. Vuibert, 2000.
- [Chen76] CHEN P.P, *The Entity-Relationship Model - Towards a Unified View of Data*, ACM Transactions on Database systems, 1-1, 1976.
- [Codd70] CODD EF, *A relational model for large shared data banks*, Communications de l'ACM, juin 1970.
- [Delmal01] DELMAL PIERRE. *SQL2 SQL3, applications à Oracle*. De Boeck Université, 2001.
- [Dupoirier95] GÉRARD DUPOIRIER, *Technologie de la GED : Techniques et management des documents électroniques*, Hermès, 1995.
- [Ford01] FORD ANDREW. *Apache précis et concis*. ISBN 2-84177-11. France : O'REILLY, 2000. 125 p. Précis et concis.
- [Gardarin99] GARDARIN GEORGES. *Bases de données : objet et relationnel*. Eyrolles, 1999.
- [Lerdof01] LERDOF RASMUS. *PHP précis et concis*. ISBN 2-84177-249-7. France : O'REILLY, 2002.
- [Mata03] MATA-TOLEDO RAMON A., CUSHMAN PAULINE K.. *Programmation SQL*. Ediscience, 2003.
- [Muller98] MULLER P.A., *Modélisation objet avec UML*, Eyrolles, 1998.
- [Pratt01] PRATT PHILIP J. . *Initiation à SQL : Cours et exercices corrigés*. Eyrolles, Collection Noire. 2001.
- [Roques04] ROQUES PASCAL, VALLÉE FRANCK. *UML 2 en action : De l'analyse des besoins à la conception J2EE*. ISBN 2-212-11462-1 (3ème édition). Paris : Eyrolles, 2004. 385 p. architecte logiciel.
- [Roques09] PASCAL ROQUES, *UML 2 par la pratique*, 7e édition, Eyrolles, 2009 [ISBN 978-2212125658]
- [Shea06] DAVE SHEA, MOLLY HOLZSCHLAG, *Le Zen des CSS*, Eyrolles, 2006.
- [Soutou02] SOUTOU CHRISTIAN. *De UML à SQL : Conception de bases de données*. Eyrolles, 2002.
- [Tardieu83] TARDIEU H., ROCHFELD A., COLLETI R., *Méthode MERISE Tome 1 : Principes et outils*, Les Editions d'Organisation, 1983.
- [Tardieu85] TARDIEU H., ROCHFELD A., COLLETI R., PANET G., VAHEE G., *Méthode MERISE Tome 2 : Démarche et pratiques*, Les Editions d'Organisation, 1985.
- [Thibaud03] THIBAUD CYRIL. *MySQL 4 : Installation, mise en oeuvre et programmation*. Editions ENI, 2003. Collection Ressources Informatiques.
- [Wong00] WONG CLINTON, *HTTP précis et concis*, O'REILLY, 2000 [ISBN 2-84177-1].

Webographie

- [w_access.developpez.com] www.developpez.com, *Les meilleurs cours, tutoriels et docs sur Access*, <http://access.developpez.com/> , consulté en 2003.
- [w_access.developpez.com/faq] www.developpez.com, *Access FAQ*, <http://access.developpez.com/faq/> , consulté en 2003.
- [w_dia] *Dia*, <http://live.gnome.org/Dia>
- [w_journaldunet.com(1)] **MORLON JÉRÔME**, *UML en 5 étapes*, <http://developpeur.journaldunet.com/dossiers/alg.uml.shtml> , 2004.
- [w_journaldunet.com(2)] **BORDERIE XAVIER**, *Cinq petits conseils pour un schéma UML efficace*, <http://developpeur.journaldunet.com/tutoriel/cpt/031013cpt.uml5conseils.shtml> , 2004.
- [w_mhubiche.developpez.com] **HUBICHE MAXENCE**, *Comprendre les jointures dans Access*, <http://mhubiche.developpez.com/Access/tutoJointures/> , consulté en 2009.
- [w_objecteering] *Objecteering software*. www.objecteering.com . [2002-septembre].
- [w_sybase] *Sybase PowerDesigner*, <http://www.sybase.com/products/enterprisemodeling> , consulté en 2002.
- [w_uml.free.fr] *UML en Français*, <http://uml.free.fr> , consulté en 2002.
- [w_univ-lyon2.fr/~jdarmont] **DARMONT JÉRÔME**. *Tutoriel SQL*. <http://eric.univ-lyon2.fr/~jdarmont/tutoriel-sql/> . [2004-janvier].
- [w_w3c.org/XML] *XML*, <http://www.w3.org/XML> .

Index

1:1	p.51, 52, 52, 53, 53	CurrentDb..... p.149, 151, 151	LCD p.10, 93, 152
1NF	p.102	Curseur p.149, 150, 150, 150, 151, 151	LDL p.10, 76, 141, 141, 142, 151
2NF	p.103	Déclaratif..... p.76	LEFT p.86
3NF	p.104, 105	Décomposition..... p.98, 102, 102, 105	Lien p.39, 40
3-tier	p.109, 111	DELETE p.81, 82, 143	LIKE p.83
Abstraite.....	p.26	Dépendance..... p.97, 98, 102, 105	Liste déroulante..... p.146
Access p.138, 140, 140, 144, 147, 149, 152, DF	p.98, 99, 99, 100, 100, 101, 101, 102	LMD p.10, 81, 82, 91, 143	
152		Diagramme..... p.16, 21, 22	Logique p.6, 8, 9, 14, 36, 36, 36, 43, 51, 51,
Administration.....	p.8, 11	Dictionnaire..... p.11	54, 65, 68, 83, 97, 98, 102, 105
Agrégat.....	p.89, 90	Différence..... p.69	Macro p.147
Algèbre p.37, 69, 69, 69, 70, 70, 71, 71, 72,	DISTINCT.....	p.82	Manipulation..... p.69
72, 73, 73		Division..... p.73	MERISE p.12, 14, 30
ALL p.92		Domaine..... p.36, 37, 37, 76, 141	Méta-langage..... p.
ALTER TABLE.....	p.79, 80	Donnée p.9	Méthode..... p.18
Analyse.....	p.11, 11, 12, 13, 13	Données..... p.9	Modèle p.6, 9, 9, 14, 14, 30, 36, 42, 42,
AND p.83		Droits p.93	97, 98, 102, 105
ANY p.93		DROP p.79	MySQL p.129
Application.....	p.140, 145, 147, 152	Dynamique..... p.26	Naturelle..... p.72
Architecture.....	p.108, 109, 111, 111, 131	E-A p.9, 12, 14, 18, 19, 30, 30, 30, 31, 31, NF	p.102
Armstrong.....	p.99, 99	32, 33, 46, 65, 65, 66, 66, 67, 67, 67, Nomalisation.....	p.97, 98
Association p.18, 22, 24, 31, 39, 40, 45, 46,	68	Normalisation p.98, 98, 99, 99, 100, 100, 101,	
47, 49, 49, 52, 52, 53, 53, 66		Enregistrement..... p.37, 38	101, 102, 102, 102, 102, 103, 104,
Association 1:1.....	p.54	Entité p.30, 32, 33, 65	105
Atomicité.....	p.102	Enumération.....	p.141 NOT p.83
Attribut p.17, 22, 32, 37, 38, 38, 39, 41, 44,	EXCEPT p.88	NOT NULL..... p.77, 78	
45, 47, 66		Objet p.118, 149, 151	
Balise p.		Occurrence..... p.9	
BCNF p.105		OCI p.129	
BD p.6, 7, 7		Fermeture..... p.100	OID p.
BETWEEN.....	p.83	Fonction..... p.89, 118, 149, 150	OMG p.16
Boucle p.118		FOREIGN KEY..... p.77, 78	On Error..... p.151
Calcul p.89		FORM p.115	Opérateur..... p.83
Cardinalité.....	p.19, 31	Formulaire..... p.115, 145, 146	Opération..... p.18, 37
Cartésien.....	p.71	Formulaires..... p.144	Optimisation..... p.8, 97, 98, 102
Catalogue.....	p.11	FROM p.82	OR p.83
Chaîne de caractères.....	p.150	Gestion d'erreur..... p.151	Oracle p.129, 131
CHECK p.77, 78		GRANT p.94	ORDER BY..... p.89
Classe p.9, 16, 21, 26, 43, 47, 118		GROUP BY..... p.90, 143	OUTER p.86
Clé p.33, 38, 39, 41, 101		HAVING..... p.90, 143	Passage..... p.43, 51, 51, 54, 65, 67, 68
Clé artificielle.....	p.39	Héritage p.20, 26, 32, 54, 54, 56, 57, 58, 59, PHP	p.116, 116, 117, 117, 118, 118, 118,
Clé candidate.....	p.38	60, 63, 64, 67, 67	119, 120, 129, 129, 129, 131
Clé primaire.....	p.38, 39	HTML p.115, 116, 116, 119, 120	Physique..... p.8, 9
Clé signifiante.....	p.39	HTTP p.111	PostgreSQL..... p.129
Client p.108, 109, 111		IF p.118	PRIMARY KEY..... p.77, 78
Codd p.36		IN p.83, 91	Problème..... p.97
Comparaison.....	p.83	INNER p.85	Produit p.37, 37, 71
Composition.....	p.24, 49	INSERT p.81, 81, 143	Programme..... p.149
Conception.....	p.6, 9, 11, 11, 97	Instance..... p.9, 10	Projection..... p.70
Conceptuel p.6, 9, 10, 11, 12, 13, 14, 16, 16, Intégrité référentielle.....		p.141	Propriété..... p.17, 22, 32
22, 30, 30, 43, 51, 51, 54, 65, 68		Interne p.10	Prototypage..... p.140
Condition.....	p.83	INTERSECT..... p.88	QBE p.143, 143
Contrainte.....	p.141	Intersection..... p.69	QueryDef..... p.149, 151, 151, 151
Contraintes.....	p.26, 51	IS NULL p.83	Question..... p.82, 91
Contrôle.....	p.8, 93	JavPHPa..... p.118	Recordset..... p.149, 150, 150, 150, 151, 151
CREATE TABLE.....	p.76	JOIN p.85, 86	Redondance..... p.7, 97, 97, 98, 102, 105
CREATE VIEW.....	p.79	Jointure p.71, 72, 72	Référence..... p.
Création.....	p.76	Langage..... p.8, 10, 81, 82, 91	REFERENCES..... p.77, 78
Critique.....	p.140	Langage de programmation..... p.149	Relation.... p.37, 37, 38, 39, 39, 40, 41, 42

<i>Relationnel</i>	p.7, 9, 14, 36, 36, 36, 37, 37, 42, <i>SGBR</i>	p.8	<i>UNION</i>	p.88
42, 43, 43, 44, 45, 45, 46, 47, 49, 49, <i>SGML</i>	p.		<i>UNIQUE</i>	p.77, 78
51, 51, 52, 52, 53, 53, 54, 54, 56, <i>Sous-requêtes</i>	p.91, 91, 92, 92, 93	<i>UPDATE</i> p.81, 81, 143
57, 58, 59, 60, 63, 64, 65, 65, 66, 66, <i>Spécifications</i>	p.13	<i>Utilisateur</i> p.93
67, 67, 67, 68, 69, 69, 73, 97, 98, 102, <i>SQL</i>	p.10, 76, 81, 82, 91, 93		<i>Variables</i> p.117
105	Table	p.76, 141, 141	<i>VBA</i>	p.149, 149, 149, 150, 150, 150, 150,
<i>Relationnel-objet</i>	p.36, 36	<i>Transaction</i> p.8
<i>Requête</i>	p.81, 82, 91, 149, 151, 151, 151	<i>Tri</i>	p.89
<i>Restriction</i>	p.70	<i>Tuple</i>	p.37
<i>REVOKE</i>	p.94	<i>Type</i>	p.9, 76
<i>RIGHT</i>	p.86		<i>Types</i>	p.141
<i>Schéma</i>	p.6, 9, 10, 42, 42	<i>UML</i> p.9, 13, 16, 16, 16, 17, 18, 20, 21, 22,
<i>Sécurité</i>	p.8, 152		22, 24, 26, 26, 43, 43, 44, 45, 45, 47,
<i>SELECT</i>	p.82, 84, 84			49, 49, 51, 51, 51, 52, 52, 53, 53, 54,
<i>Serveur</i>	p.108, 109, 111, 111, 116			54, 56, 57, 58, 59, 60, 63, 64, 68
<i>SGBD</i>	p.6, 7, 7, 7, 9		<i>Union</i>	p.69