Module 4 - Modélisation: associations et normalisation

Dans ce module, nous allons voir comment créer des *associations* entre relations (tables), comment modéliser ces associations. Nous verrons également les grandes étapes de la conception d'une base de données.

Nous aborderons également le sujet important qu'est la *normalisation* des données, ainsi que l'usage de `NULL`. Enfin, nous apprendrons à interroger les données (DQL) avec les *fonctions*, *opérateurs ensemblistes* et *de regroupement*.

Enfin, nous réaliserons notre première *jointure*.

Les grandes étapes de la conception

On ne conçoit pas une base de données directement en SQL. On divise le processus en 4 étapes:

- niveau conceptuel
- niveau relationnel
- niveau physique (ou SQL)
- niveau externe

> Schéma extrait de *Modélisation des bases de données 4e édition, C. Soutou, Eyrolles, 2017*

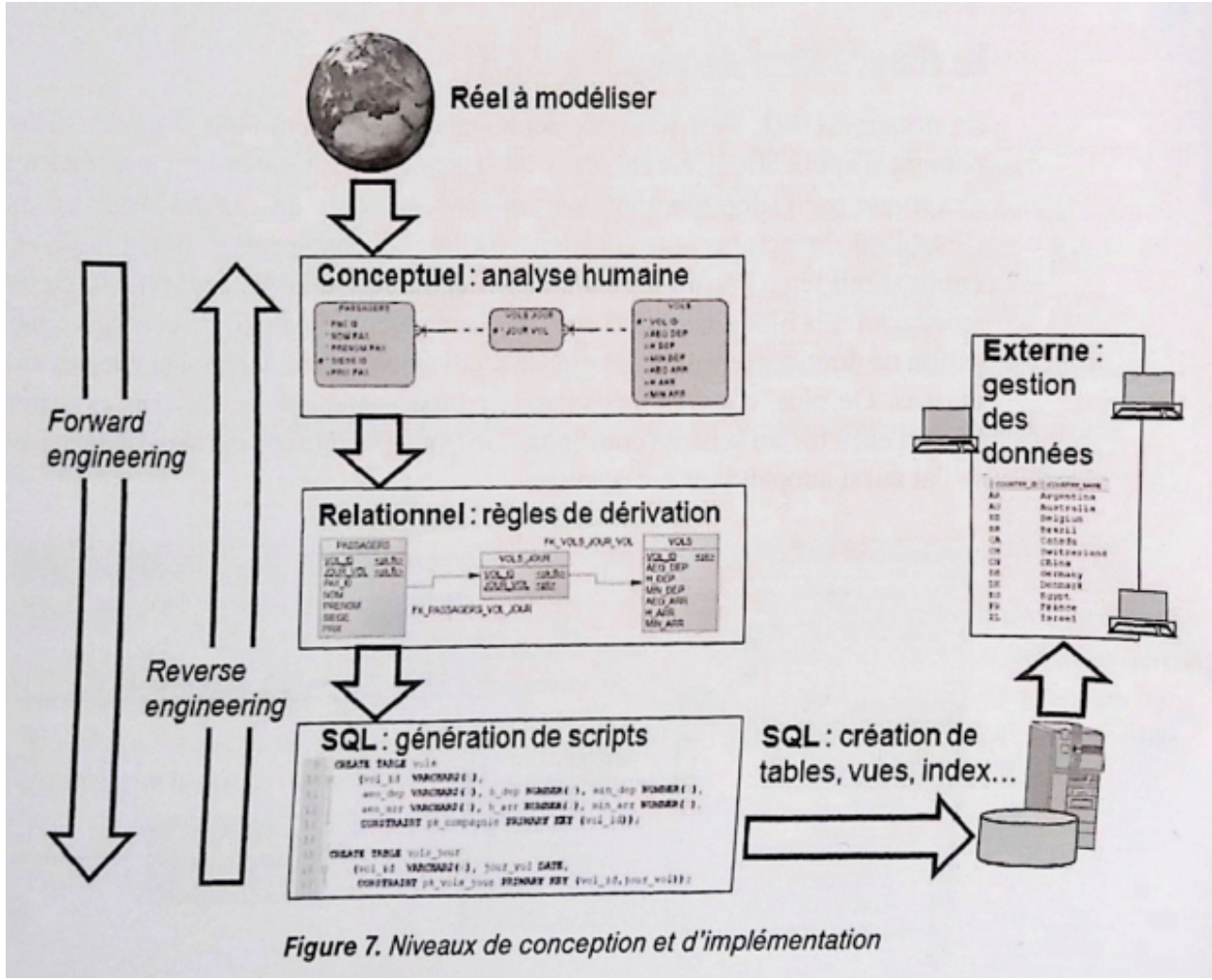


Figure 7. Niveaux de conception et d'implémentation

Le niveau *conceptuel* (MCD)

Le niveau conceptuel des données (MCD) décrit les entités du monde réel, en termes d'objets, de propriétés et de relations, *indépendamment de toute technique d'organisation et d'implantation des données*.

Ce modèle se concrétise par des schémas **UML** ou entités-associations représentant la structure du système d'information, *du point de vue des données*.

On y commencera également le dictionnaire des données, le cahier des charges en partant du recueil des besoins.

Le niveau *relationnel*

Le niveau relationnel est dérivé (déduit) du niveau conceptuel. C'est une étape de *traduction* où l'on applique généralement des méthodes bien définies et connues (notamment pour traiter le cas des associations).

On produira ici des diagrammes UML représentant cette fois le modèle relationnel (clefs, associations sous forme de références, types SQL, etc.)

Le niveau *physique* (SQL)

Implémentation du modèle relationnel sous forme de script SQL. Écriture du code SQL. **Choix d'une implémentation** (d'un SGBD comme MySQL, Oracle, SQL Server, PostgreSQL, etc.).

Écriture des procédures stockées, déclencheurs d'évènements, etc et création des index.

Le niveau *externe* (les Vues)

Création des interfaces de la base pour les utilisateur·ices de la base de données (vues, accès)

Associations

Jusqu'ici nous avons décrit et utilisé les relations.

L'intérêt du modèle relationnel réside dans le fait de **découpler le stockage des informations de l'utilisation de ces informations** (pas nécessairement connu à l'avance).

Pour cela, on découpe notre modèle en créant des relations *associées* entre elles. **L'utilisation des données se fera en faisant des jointures entre ces relations**.

Associations

Les associations permettent de:

- faire référence à une donnée déjà existante dans la base
- éviter la redondance des données
- modélisation métier qui a du *sens*

Niveau conceptuel : modéliser les associations

Pour modéliser les relations et leurs associations (liens entre elles), on va utiliser ici le formalisme UML.

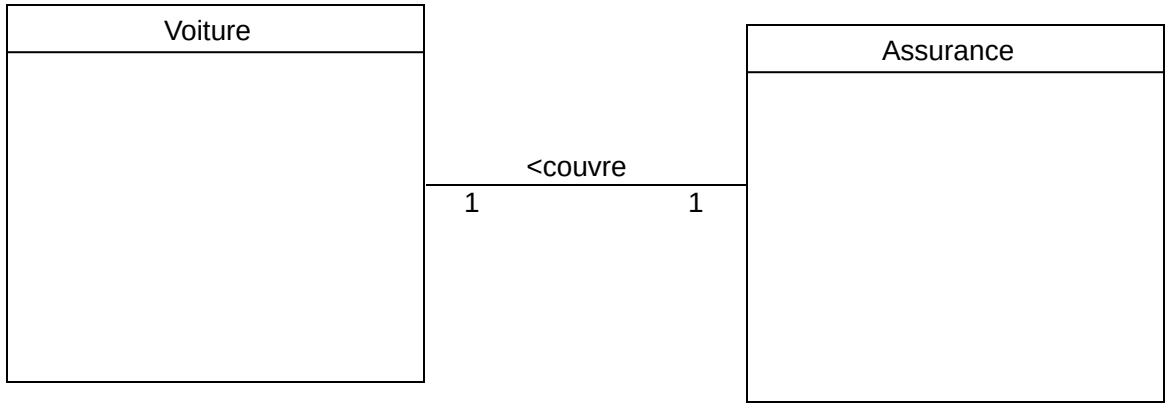
> On aurait pu également utiliser d'autres formalismes comme le modèle entité-association de Merise, toujours très utilisé en conception de base de données.

Les différents types d'association

> En SQL, cela se traduit par *Comment une ligne de ma table est reliée à d'autres lignes de la même ou d'une autre table ?*

- association **un-à-un** (*one-to-one*) Ex: un contrat d'assurance ne concerne qu'une voiture
- association **un-à plusieurs** (*one-to-many*) Ex: un pilote travaille pour une compagnie, une compagnie fait travailler plusieurs pilotes
- association **plusieurs-à plusieurs** (*many-to-many*) Ex: un panier d'achats contient plusieurs produits, un produit peut se trouver dans plusieurs paniers d'achats

Association un-à-un (*one-to-one*)



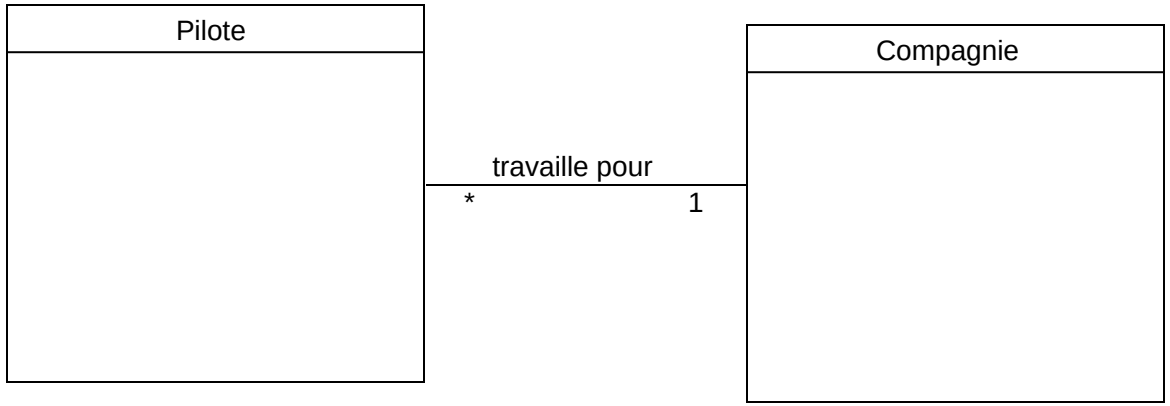
Association la moins courante.

Cette relation relie un élément d'une relation à un autre (et un seul)

Par exemple, un contrat d'assurance concerne une voiture, une voiture ne peut être assurée que par un contrat en même temps.

> L'autre élément peut appartenir à une autre ou à la même relation (association dite *réflexive*) !

Association un-à plusieurs (*one-to-many*)

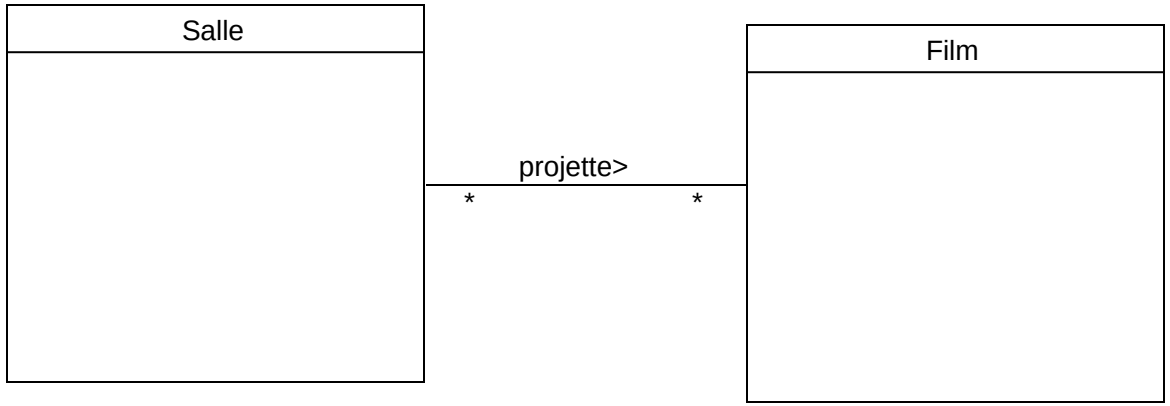


Association la plus courante et omniprésente dans le monde. Reflète notamment le concept de *hiérarchie*.

Cette relation relie un élément d'une relation à 0, un ou plusieurs autres.

Par exemple, un pilote travaille pour une compagnie (contrat d'exclusivité), une compagnie embauche plusieurs pilotes.

Association plusieurs-à plusieurs (*many-to-many*)

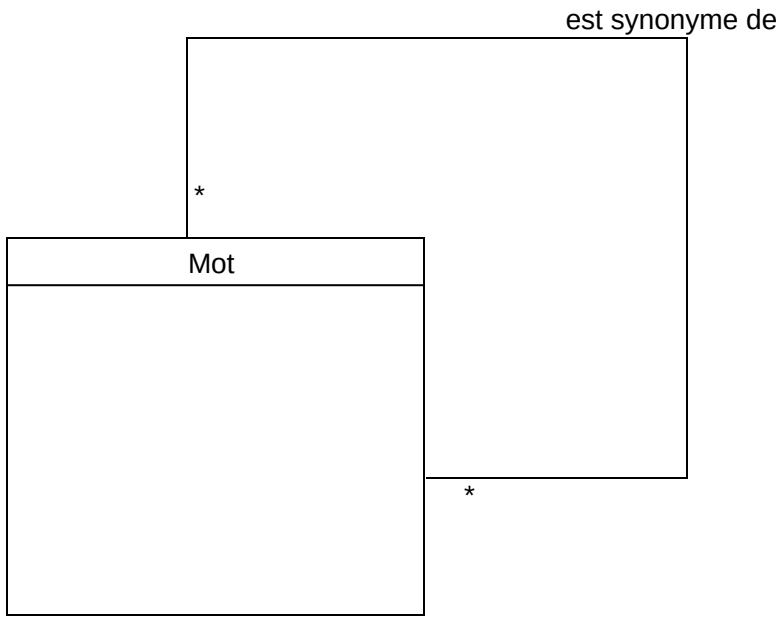


Association courante.

Dans cette association, chaque élément peut être relié à 0, un ou plusieurs éléments de l'autre relation.

Par exemple, dans un cinéma, une salle peut projeter plusieurs films et un film peut être projeté dans plusieurs salles.

Le cas des associations réflexives



Une association *réflexive* est une association qui relie une relation à elle-même.

> Nous n'avons pas le temps d'explorer toutes les associations réflexives. Pour en savoir plus, lisez ce très bon article de E. Thirion sur le sujet (attention, le formalisme de Merise est utilisé !)

Plusieurs associations entre deux relations

Plusieurs associations peuvent exister entre deux relations.

Par exemple, un train peut être modélisé par trois associations avec une gare:

- gare de départ
- gare d'arrivée
- gare desservie sur le trajet

Définition d'une association

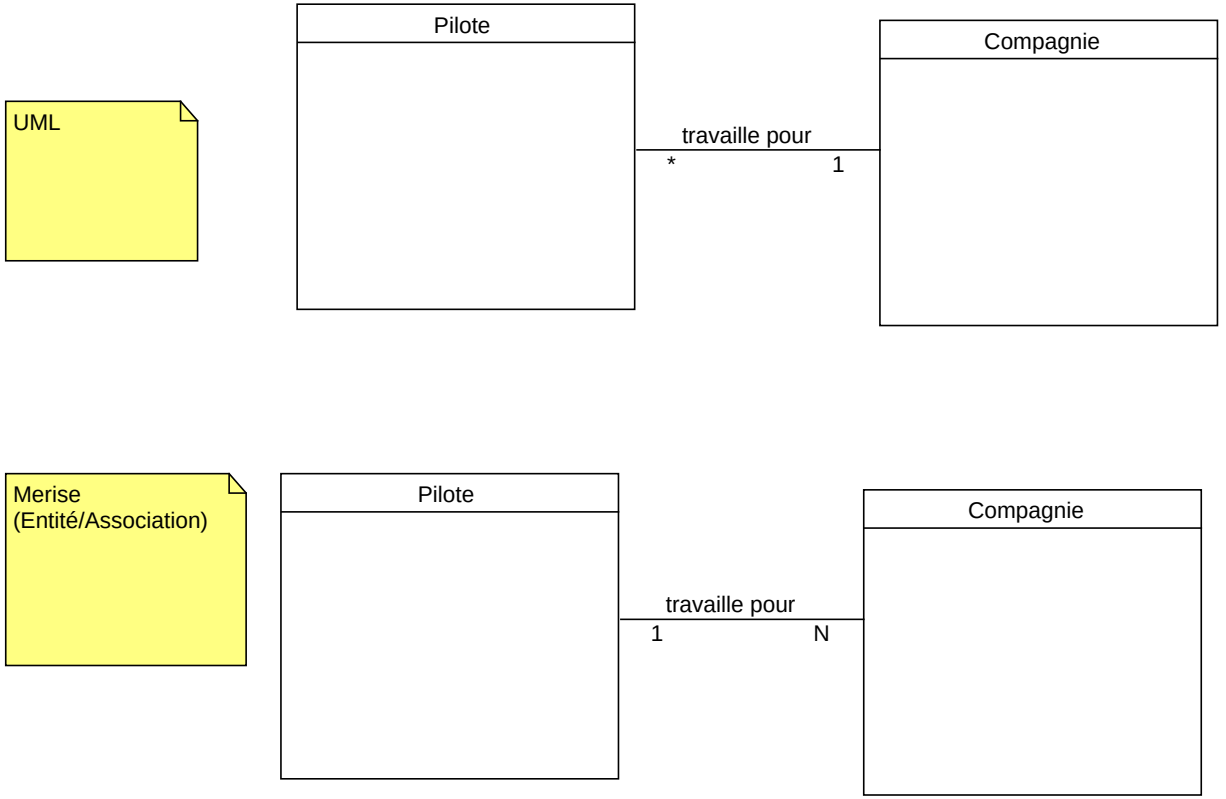
Une association est définie par:

- une multiplicité (ou cardinalité en Merise): indique le nombre d'éléments avec lequel l'élément situé à l'autre extrémité de l'association est relié
- un nom (un verbe)

Notation: UML vs Merise (basé sur le formalisme Entité/Association)

Attention, ces deux formalismes ont des conventions différentes. La position de la multiplicité sur le dessin de l'association **est inversée entre les deux formalismes** !

> Le modèle entité-association à d'autres particularités graphiques non représentées ici.



Notation: UML vs Merise (basé sur le formalisme Entité/Association)

Définition	UML (Multiplicité)	Merise (Cardinalité)
Associé à zéro ou un élément	0..1	0,1
Associé à un et un seul élément	1..1 (ou 1)	1,1
Associé à zéro ou plusieurs éléments	0..* (ou *)	0,N (ou 0,n)
Associé à 1 ou plusieurs éléments	1..*	1,N (ou 1,n)

> Remarque: le terme **cardinalité** est meilleur et bien défini en mathématiques.

Niveau relationnel: passage du modèle conceptuel au modèle relationnel

Passage du modèle conceptuel au modèle relationnel: association *un-à plusieurs*

Cette association se résout à l'aide d'une contrainte de clef étrangère. La clef étrangère se place toujours du côté *plusieurs*. Par exemple, sur l'association `Client--*Facture`, la clef étrangère se place sur la relation `Facture`. Elle contiendra alors un attribut **de même type** que la clef primaire de la relation `Client`, qui y fera référence.

Passage du modèle conceptuel au modèle relationnel: association *plusieurs-à plusieurs*

Il est **impossible** de modéliser cette association dans le modèle relationnel **sans introduire une nouvelle relation** !

Cette association se résout à l'aide d'une *table de jointure*. Cette table comporte (au moins) deux colonnes, chacune portant une contrainte de clef étrangère vers chaque clef primaire des relations impliquées. La clef primaire de la table de jointure est **composée** de ces deux colonnes.

> Une table de jointure peut comporter d'autres attributs, cf l'exemple avec le SI d'un cinéma.

Passage du modèle conceptuel au modèle relationnel: association *un-à-un*

Comme pour l'association *un-à plusieurs* (dont l'association *un-à-un* est un cas particulier), cette association se résout à l'aide d'une contrainte de clef étrangère. La clef étrangère se place indifféremment sur une relation ou sur l'autre.

Normalisation et usage de `NULL`

Normalisation des données

> La normalisation des données est un concept important.

La normalisation est un processus d'organisation des données tel que:

- les redondances soient évitées (moins d'espace disque nécessaire, *moins de risques d'incohérences*). L'utilisation de clé étrangère participe à la normalisation
- la programmation de n'importe quelle opération sur les tables (ajout, modification, suppression) soit facilitée.* Si c'est dur de mettre à jour une donnée de manière cohérente, c'est qu'il y a un souci de normalisation.*
- l'intégrité (exactitude) des données soit renforcée

Normalisation des données

Bien normaliser amène à un grand nombre de petites tables (avec peu d'attributs/colonnes) et à l'utilisation intensive de jointures.

> Et c'est ok ! C'est fait pour.

Pour en savoir plus, regarder les formes normales, le standard défini pour la normalisation.

Sur l'usage de `NULL`

> `NULL`, c'est nul !

- inflige de gros dégâts en termes de performances et en place allouée. Une ligne qui contient `NULL` alloue quand même un espace mémoire (inutilement)
- pour comparer `NULL` à autre chose (dans une clause `WHERE` ou dans une jointure) il faut ajouter des règles supplémentaires.
- ruine les performances des index
- oblige à écrire des requêtes plus complexes en DQL (tester si une donnée est `NULL` avec `IS NULL`)

Sur l'usage de `NULL`

D'accord, mais si je n'ai pas forcément de valeur pour une colonne ? **Placer cette colonne dans une relation externe** (une nouvelle table !). `NULL` se transforme en *pas d'enregistrement*.

Par exemple, prenons une relation `Personne` qui possède un attribut `numero_telephone`. Tout le monde n'a pas forcément un numéro de téléphone. Donc au lieu de mettre une colonne `numero_telephone` (possiblement `NULL`), **créer** une relation `Telephone` avec une association *un-à-plusieurs* entre `Personne` et `Telephone`.

Les fondamentaux de la modélisation

Pour résumer quelques fondamentaux à suivre

- pas de `NULL`
- données atomiques (qui ne peuvent pas être divisées en d'autres données plus petites)
- pas de redondance (normalisation)
- la modification d'une information ne doit **impacter qu'une ligne**

> C'est un guide, à suivre au mieux et savoir pour quelles raisons on s'en écarte.

Ressources

- Conception des bases de données (modèle Entité-Association), très bonne documentation sur les relations *n-aires* et sur la modélisation des bases de données de manière générale. *Attention, les cardinalités dans le modèle Entité-Association (utilisé par Merise notamment) sont **inversées par rapport aux cardinalités du modèle UML** (ce serait trop simple sinon...)*
- Modélisation des bases de données: UML et les modèles entité-association, 4e édition ou 5e édition, de Christian Soutou et Frédéric Brouard, 2022 (pour la 5e). Une très bonne référence sur la modélisation des bases de données relationnelles, du modèle conceptuel au niveau physique. Très complet, plein de conseils, et d'exercices inspirés de cas réels. Ces deux auteurs ont souvent travaillé ensemble et le résultat est là (théorie, pratique, expertise en milieu pro). Un *must have*. (Pas fait pour les débutants sur le modèle relationnel, à étudier/parcourir après avoir acquis les bases et pratiquer un peu)

Ressources

- UMLet, un logiciel gratuit et open source pour dessiner rapidement des diagrammes UML et relationnels et les exporter. Développé en Java donc cross-platform. **Parfait pour du dessin rapide et du partage** (chaque diagramme est enregistré sous la forme d'un code source). **Ne permet pas de se servir des diagrammes pour générer du code (langage POO, SQL)**.
- UMLet extension VS code, UMLet peut être intégré directement dans VS Code. Il existe également un plugin pour Eclipse