الجـــمهوريـــة الجزائريـــة الديمقراطيـــة الشعبيـــة

وزارة التعليم العالـــي و البحث العلمـــي

المدرسة العليا لعلـــوم التسييـــر

Ecole Supérieure des Sciences de Gestion ANNABA

Intitulé du Polycopié

Les bases de données pour débutant

Module : Informatique 2 - Les base de données

Classes : Préparatoires

Niveau : Deuxième Année

Polycopié Réalisé Par :

Anis BEY

Année Universitaire

2020/2021

Sommaire

Préface

Ce cours vise à introduire les principes des bases de données informatique aux étudiants en classe préparatoires dans les écoles supérieures des sciences de gestion conformément au programme notional. Après la première année préparatoire où les étudiants auraient appris les bases de l'informatique ainsi que les fondements de la programmation, ce cours constitue une continuité de la base de la programmation informatique qui traite la gestion des données dans les systèmes.

Les connaissances et les compétences visées à travers ce cours sont les suivantes :

- Qu'est ce qu'une base de donnée ?
- Quand est ce qu'on utilise les bases de données ?
- Qu'est ce qu'un système de gestion de bases de données (SGBD) ?
- Le processus de développement d'une base de données
    - Savoir analyser l'univers d'application
    - Savoir élaborer un modèle entité-association
    - Savoir passer d'un modèle entité-association à un modèle relationnel
    - Savoir écrire des requêtes SQL
    - Savoir implémenter le modèle relationnel en utilisant un SGBD cible

I Introduction

Aujourd'hui, les bases de données ont pris une place essentielle dans l'informatique, plus particulièrement en gestion des données. Au cours des trente dernières années, des concepts, méthodes et algorithmes ont été développés pour gérer des données. Ce qui constitue aujourd'hui l'essentiel de la discipline Bases de Données (BD). Cette discipline est utilisée dans de nombreuses applications. Il existe un grand nombre de Systèmes de Gestion de Bases de Données (SGBD) qui permettent de gérer efficacement de grandes bases de données. De plus, une théorie fondamentale sur les techniques de modélisation des données et les algorithmes de traitement a vu le jour. Les bases de données constituent donc une discipline s'appuyant sur une théorie solide et offrant de nombreux débouchés pratiques.

Ce cours se divise essentiellement en quatre sections. Dans la première section, des définitions de l'essentiel des bases de données ainsi que tout les objets liés à ces dernières sont exposés. La deuxième parties présente le processus de conception d'une base de données bien détaillé ainsi que le language SQL, qui est un language de manipulation des bases de données. Dans les deux dernière parties, une introduction à ACCESS et quelques exercices sont aussi présentés pour bien illustrer les notions théoriques vues dans les premières sections.

1. Définition d'une base de données

Une base de données (BD), en anglais Data Base (DB), est une collection d'informations structurées, cohérentes et persistantes qui modélise un objet du monde réel.

Autrement dit, c'est une collection de données dont la structure reflète les relations qui existent entre ces données. Cet aspect n'apparaît pas dans la notion de fichier vu en cours de programmation. Comme vous l'avez pu constater dans la représentation des données dans les fichiers, les données des fichiers sont décrites dans les programmes tandis que les données de la BD sont décrites hors des programmes dans la base elle-même. La multiplication des fichiers entraînait la redondance des données, ce qui rendait difficile les mises à jour. D'où l'idée d'intégration et de partage des données.

2. Définition d'un SGBD

Un système de gestion des bases de données (SGBD) est un système informatique à travers lequel on peut créer et gérer des bases de données. Un SGBD peut donc être défini comme un ensemble de logiciels systèmes permettant de stocker et d'interroger un ensemble de fichiers interdépendants, mais aussi comme un outil permettant de modéliser et de gérer les données.

Un SGBD est l'ensemble de procédures permettant :

- La description des données et des relations les concernant,
- L'interrogation de la base,
- La mise à jour,
- Le partage des données,
- La protection des données de la base.

Chaque SGBD possède ses propres spécificités et caractéristiques. La syntaxe aussi est parfois différentes d'un SGBD à un autre. Parmi les SGBD les plus utilisés on peut citer : MySQL, PostgreSQL, SQLite, Microsoft SQL Server ou encore Oracle.

3. Historique des SGBD et des bases de données

L'histoire des bases de données remonte aux années 1960, avec l'apparition des bases de données réseau et des bases de données hiérarchiques. Dans les années 1980, ce sont les bases de données object-oriented qui ont fait leur apparition. Aujourd'hui, les bases de données prédominantes sont les SQL, NoSQL et bases de données cloud.

Il est aussi possible de classer les bases de données en fonction de leur contenu : bibliographique, textes, nombres ou images. Toutefois, en informatique, on classe généralement les bases de données en fonction de leur approche organisationnelle. Il existe de nombreux types de bases de données différentes : **relationnelle, distribuée, cloud, NoSQL**…

D'une manière chronologique, voici les grandes dates marquantes dans l'histoire des bases de données.

50 - 60 Fichiers et méthodes d'accès (séquentiel, direct, séquentiel indexé).

62 - 63 Apparition du concept de Base de Données.

65 - 70 Conception des SGBD de 1ère Génération (modèles hiérarchique et réseau)

70 - 85 2ème Génération des SGBD organisés sur le modèle relationnel.

Systèmes commercialisés dans les années 1980 :

- MRDS de Honeywell diffusé par CII-HB,
- QBE (Query By Example),
- SQL/IDS d'IBM,
- INGRES de Relational Technology,
- ORACLE de Relational Software.

4. Les types de bases de données

Dans le cas d'une grande database, les multiples utilisateurs doivent être en mesure de manipuler les informations qu'elle contient rapidement et n'importe quand. De plus, les grandes entreprises ont tendance à cumuler de nombreux fichiers indépendants comprenant des fichiers liés ou même des données qui se superposent. Dans le cadre d'une représentation de données, il est nécessaire que les données en provenance de plusieurs fichiers puissent être liées. C'est pourquoi différents types de bases de données ont été développés pour répondre à ces exigences : orientée texte, hiérarchique, réseau, relationnelle, orientée objet…

Base de données hiérarchique

Les bases de données hiérarchiques comptent parmi les plus anciennes bases de données. Au sein de cette catégorie, les enregistrements sont organisés dans une structure d'arborescence. Chaque niveau d'enregistrements découle sur un ensemble de catégories plus petites.

Base de données réseau

Les bases de données réseau sont également parmi les plus anciennes. Plutôt que de proposer des liens uniques entre différents ensembles de données à divers niveaux, les bases de données réseaux créent des liens multiples entre les ensembles en plaçant des liens, ou des pointeurs, sur un

ensemble d'enregistrements ou un autre. La vitesse et la polyvalence des bases de données réseau ont conduit à une adoption massive de ce type de databases au sein des entreprises ou dans le domaine du e-commerce.

Base de données orientée texte

Une database orientée texte, ou flat file database, se présente sous la forme d'un fichier (une table) au format .txt ou .ini. Un fichier plat est un fichier texte, ou un fichier combinant du texte avec un fichier binaire. En général, dans ces bases de données, chaque ligne ne comporte qu'un enregistrement.La plupart des bases de données pour PC sont des bases de données orientées texte.

Base de données SQL (relationnelle)

Les bases de données relationnelles ont été inventées en 1970 par E.F. Codd de IBM. Il s'agit de documents tabulaires dans laquelle les données sont définies afin d'être accessibles et de pouvoir être réorganisées de différentes manières.

Les bases de données relationnelles sont constituées d'un ensemble de tableaux. Au sein de ces tableaux, les données sont classées par catégorie. Chaque tableau comporte au moins une colonne correspondant à une catégorie. Chaque colonne comporte un certain nombre de données correspondant à cette catégorie.

L'API standard pour les bases de données relationnelles est le Structured Query Language (SQL). Les bases de données relationnelles sont facilement extensibles, et de nouvelles catégories de données peuvent être ajoutées après la création de la database originale sans avoir besoin de modifier toutes les applications existantes.

Base de données distribuée

Une BDD distribuée est une database dont certaines portions sont stockées à plusieurs endroits physiques. Le traitement est réparti ou répliqué entre différents points d'un réseau.

Les bases de données distribuées peuvent être homogènes ou hétérogènes. Dans le cas d'un système de base de données distribuée homogène, tous les emplacements physiques fonctionnent avec le même hardware et tournent sous le même système d'exploitation et les mêmes applications de bases de données. Au contraire, dans le cas d'une database distribuée hétérogène, le hardware, les

systèmes d'exploitation et les applications de bases de données peuvent varier entre les différents endroits physiques.

Base de données cloud

Dans ce cadre, elle est optimisée ou directement créée pour les environnements virtualisés. Il peut s'agir d'un cloud privé, d'un cloud public ou d'un cloud hybride.

Les bases de données cloud offrent plusieurs avantages comme la possibilité de payer pour la capacité de stockage et la bande passante en fonction de l'usage. Par ailleurs, il est possible de changer l'échelle sur demande. Ces bases de données offrent aussi une disponibilité plus élevée.

Base de données NoSQL

Les bases de données NoSQL sont utiles pour les larges ensembles de données distribuées. En effet, les bases de données relationnelles ne sont pas conçues pour le Big Data, et les ensembles de données trop larges peuvent poser des problèmes de performances.

Si une entreprise doit analyser d'importantes quantités de données non structurées, ou des données stockées sur plusieurs serveurs cloud virtuels, la database NoSQL est idéale. Avec l'essor du Big Data, les bases de données NoSQL sont de plus en plus utilisées.

Base de données orientée objets

Les objets créés à l'aide de langage de programmation orientés objets sont généralement stockés sur des bases de données relationnelles. Toutefois, en réalité, les bases de données orientées objets sont plus adaptées pour stocker ce type de contenu.

Plutôt que d'être organisée autour d'actions, les bases de données orientées objets sont organisées autour d'objets. De même, au lieu d'être organisées autour d'une logique, elles sont organisées autour des données. Par exemple, un enregistrement multimédia au sein d'une BDD relationnelle peut être défini comme un objet de données plutôt que comme une valeur alphanumérique.

Base de données orientée graph

Une base de données orientée graphe, ou graphe, est un type de database NoSQL utilisant la théorie des graphes pour stocker, cartographier et effectuer des requêtes sur les relations entre les données. Les bases de données graphe sont constituées de noeuds et de bords.

Chaque noeud représente une entité, et chaque bord représente une connexion entre les noeuds. Les bases de données graphes gagnent en popularité dans le domaine des analyses d'interconnexions. Par exemple, les entreprises peuvent utiliser une BDD graphe pour miner des données sur ses clients à partir des réseaux sociaux.

De plus en plus souvent, des bases de données jadis séparées sont combinées électroniquement sous forme de collections plus larges que l'on appelle les Data Warehouses. Les entreprises et les gouvernements utilisent ensuite des logiciels de Data Mining pour analyser les différents aspects des données. Par exemple, une agence gouvernementale peut procéder ainsi pour enquêter sur une entreprise ou une personne qui ont acheté une grande quantité d'équipement, même si les achats sont disséminés dans tout le pays ou répartis entre plusieurs subsidiaires.

5. Caractéristiques des SGBD

Les principales caractéristiques d'un système de gestion des bases de données sont les suivantes :

- **Indépendance physique (données/programmes)**

C'est la possibilité de modifier l'organisation physique (accès) sans modifier les programmes.

- **Indépendance logique**

Consiste à la modification du schéma conceptuel sans modification des programmes.

- **Manipulation des données**

La possibilité de la manipulation des données par des utilisateurs qui n'ont pas la connaissance de l'organisation de la base et qui disposent de langages évolués « naturels ».

- **Efficacité des accès aux données**

Offrir la possibilité aux utilisateurs de manipuler les données à partir de langages hôtes (Pascal, Fortran, C, Java…) tout en assurant une efficacité et rapidité au niveau des accès sur les supports.

- **Administration centralisée des données**

L'administrateur de la base définit les structures de données, de stockage et de contrôle.

- **Intégrité des données**

Pouvoir assurer une cohérence des données lors des mises à jour (les règles de contraintes d'intégrité sont définies par l'administrateur).

- **Partageabilité des données**

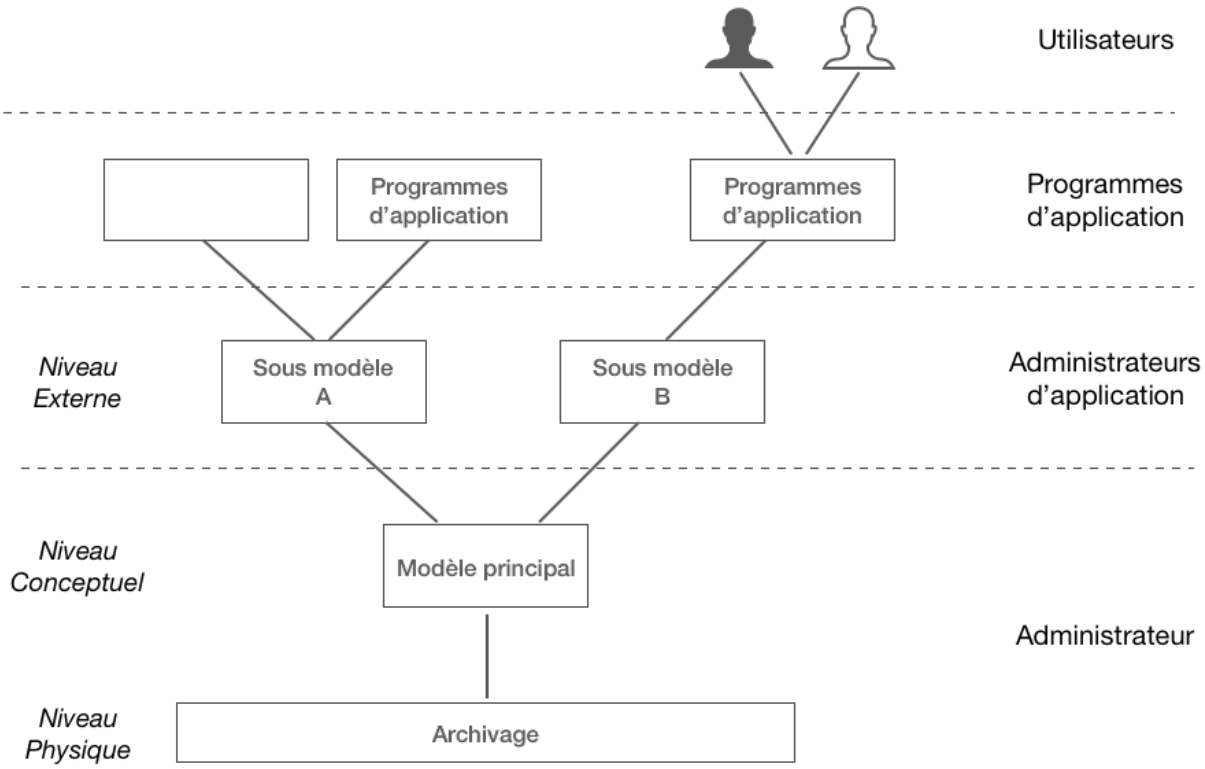
Permettre à plusieurs applications l'accès et l'exploitation simultané des données.

- **Sécurité des données**

Pouvoir donner la possibilité de contrôle des droits d'accès et la facilité de la reprise sur panne.

6. Architecture d'un SGBD

La figure suivante montre le fonctionnement d'un système de gestion de bases de données dans une application informatique ainsi que les différents utilisateurs et leurs rôles.



- **Niveau interne ou physique**

Ce niveau représente le stockage physique des données (fichiers, disques, etc.) et des méthodes d'accès (index, chaînages, etc.).

- **Niveau externe**

Ce niveau représente la vue externe pour chaque groupe d'utilisateurs sur un sous ensemble de la base. Un schéma externe est généralement un sous schéma du schéma conceptuel mais peut contenir parfois des informations supplémentaires spécifiques à l'implémentation dans le SGBD.

7. Les différents utilisateurs d'une base de donnée

- **L'administrateur principal de la base**

Il définit le schéma conceptuel. Cette opération conditionne l'évolution de la base. Aussi il définit les modalités de protection des données.

- **Les administrateurs d'applications**

Ils définissent le sous modèle adapté à l'application, l'élaboration des schémas externes, la définition des règles de correspondance entre externe et conceptuel.

Le travail des administrateurs conduit à l'élaboration du dictionnaire des données : ensemble de schémas et règles de correspondance entre schémas associés à la base de données.

- **Programmeurs d'application**

Leur rôle est d'établir des bibliothèques de programmes de manipulation et de traitement de la base (interrogation, mise à jour, etc.). Utilisation d'un Langage de Manipulation de Données (LMD) qui peut être autonome ou associé à un langage hôte et qui dispose des caractéristiques des langages algorithmiques.

II Conception de base de données

La conception de la base est l'étape la plus importante dans le cycle de développement de n'importe quelle base de donnée. Cette étape qui précède l'implémentation avec un SGBD consiste à l'élaboration d'un schéma clair et précis afin qu'il puisse être implémenté.

Concevoir la base de données nécessite une première phase de modélisation conceptuelle qui consiste à déterminer quelles sont les structures de données pertinentes et les relations qui existent entre ces dernières. Nous traitons ici la modélisation conceptuelle des données : Comment élaborer un schéma de bases de données?

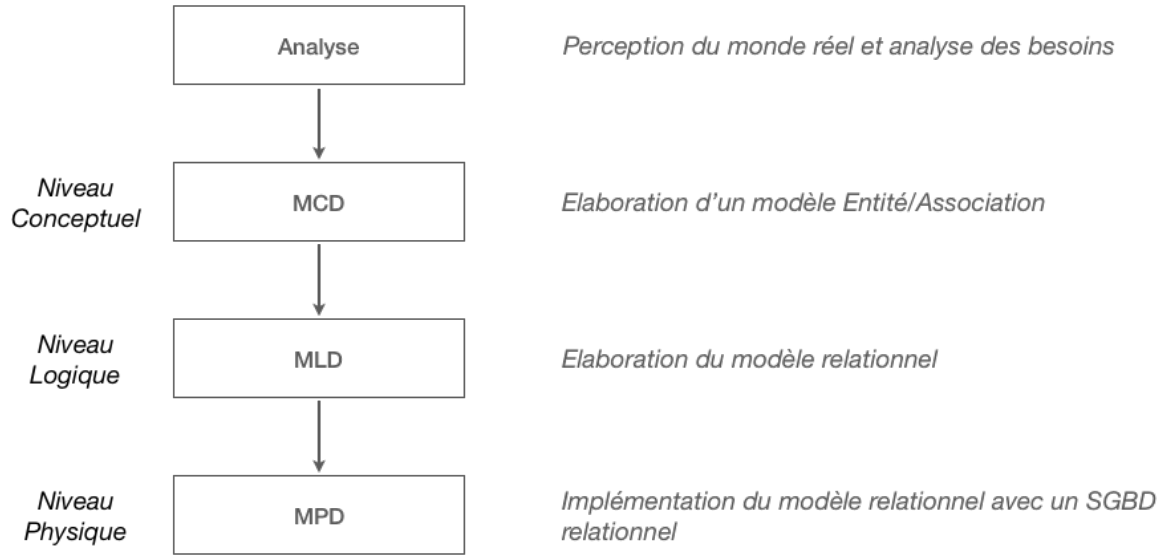
Pas de système d'information correct sans modélisation ! Pour cela il existe des concepts formels qui viennent du monde des bases de données. Dans ce cours, nous introduisons les concepts généraux ainsi que la terminologie appropriée.

La phase de conception de la BD est une phase de réflexion sur la structure des données en fonction des besoins de l'application: données importantes, propriétés, contraintes, requêtes à prévoir... en

accord avec les utilisateurs. Conceptuel signifie qu'on est indépendant des solutions informatiques. L'intérêt d'établir un schéma conceptuel réside dans le fait d'être accès sur une application, d'être indépendant des technologies donc portable et facilitant l'échange d'informations, établi selon un modèle formel sur des spécifications non ambiguës.

La modélisation conceptuelle des données, c'est l'activité d'élaboration du schéma conceptuel selon un modèle conceptuel. Il s'agit de poser sur le papier le schéma conceptuel qui permettra d'établir la structure des données (schéma logique) dans le modèle logique : sous forme de tables dans le modèle relationnel ou de classes dans le modèle orienté objet.

Un schéma c'est l'expression de la description de la base de données obtenue en employant un modèle de données. Un modèle conceptuel est un cadre formel pour schématiser le contenu des informations selon un formalisme établi, graphique si possible. Nous verrons dans la section suivante le modèle Entité/Association.



Modèle conceptuel de données

Un modèle conceptuel de données est un ensemble de concepts qui permettent de décrire et de manipuler des données du monde réel, et de règles d'utilisation de ces concepts. Les modèles comportent 2 parties : une partie statique qui décrit la structure des données (MCD) et les contraintes explicites sur ces données (CI), et une partie dynamique qui définit les traitements sur les données (MCT).

Les concepts de base de la modélisation sont :

- Les objets regroupés en classes et identifiés,

- Les liens entre objets avec leurs cardinalités,

- Les propriétés des objets,

- La représentation multiple des objets.

Un modèle conceptuel doit respecter les propriétés suivantes :

- Complétude (Description de tous phénomènes courants nécessaires à l'application)

- Fiabilité (formellement défini)

- Orientation utilisateur (compréhensible, clair, lisible)

- Orthogonalité (les concepts proposés doivent être indépendants)

- Compatibilité logiciel (traduisible en SGBD existant)

- Complètement opérationnel (capacités de manipulation des données)

1. Processus de conception d'une bases de données

Une BD est une représentation de la partie du monde réel qui nous intéresse. Lors de la conception d'une BD pour les besoins d'un utilisateur, après interview, l'objectif est d'élaborer le schéma conceptuel de son application. L'utilisateur a une perception du monde réel axée sur son application : chaque utilisateur a sa propre focale d'observation. Son analyse de la réalité est donc partielle (elle ne représente que les informations intéressantes pour son application), subjective (elle représente le point de vue du concepteur) et infidèle (ne représente pas la réalité telle qu'elle est, mais telle qu'elle intéresse le concepteur). Les phénomènes observés sont abstraits en classes, puis représentés et décrits dans un schéma conceptuel selon le modèle choisi. Il important de rappeler que nous pouvons obtenir plusieurs schémas conceptuels pour les mêmes phénomènes du monde réel observés. Cependant le schéma conceptuel obtenu doit être conforme au modèle conceptuel choisi.

2. Le modèle Entité Association

Dans le modèle EA (Entité/Association) ou ER (Entité/Relation), les objets sont représentés par des entités, et les liens par des associations, aussi appelées relations. Nous voyons ici les concepts qui vont nous permettre d'élaborer des schémas conceptuels EA.

2.1. Entités

Une entité E est la représentation d'un objet du monde réel (concret ou abstrait) perçu par le concepteur comme ayant une existence propre, et à propos duquel on veut enregistrer des informations. Une entité existe indépendamment du fait qu'elle puisse être liée à d'autres entités de la BD.

Notre schéma conceptuel EA comprend deux entités : PERSONNE et VOITURE et une relation (ou association) POSSEDE. Un exemple d'entité PERSONNE est Mr Benmohamed. Une entité du VOITURE est la voiture immatriculé sous le numéro 30345-00-16.

Le formalisme graphique proposé ici présente les entités sous forme de rectangle. On trouvera cependant dans la littérature de nombreux formalismes. Dans notre cas, on va utiliser le rectangle pour les entités et l'oval pour les relations. Il est donc important de préciser en cas d'ambiguïté si vous en changez.

2.2. Associations/Relations

Une association A est la représentation d'un lien non dirigé entre plusieurs entités (qui jouent un rôle déterminé). Les entités PERSONNE et VOITURE sont liés par une relation POSSEDE.

Rôles

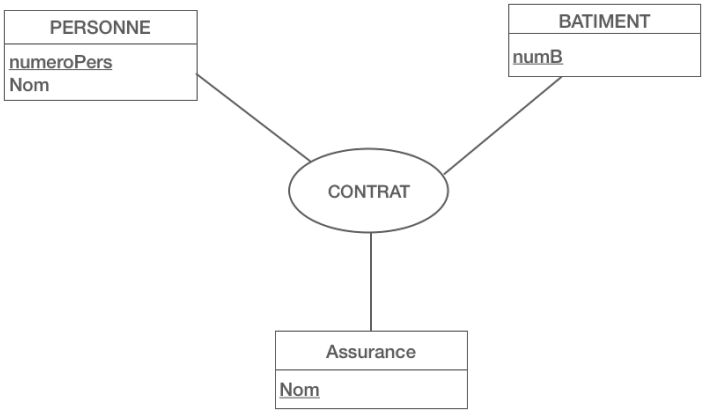
Une association a deux rôles de part et d'autre de l'association pour chacun des entités. Les rôles peuvent être clairement explicités sur le schéma conceptuel.

- Le rôle *possède* : Mr. Benmohamed possède la voiture 30345-00-16.

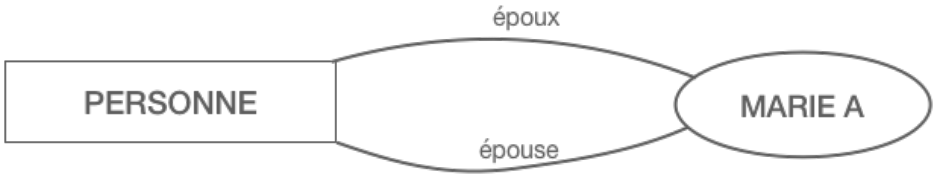
- Le rôle *est possédé par*: La voiture 30345-00-16 est possédé par Mr. Benmohamed.

Les relations ternaires

Les relations ou associations sont la plupart du temps binaires, impliquant l'association de deux entités. Ils peuvent également être ternaires en mettant en jeu l'association de 3 entités, quaternaires en permettant l'association de 4 entités, etc. Dans l'exemple ci-dessous, une occurrence de l'entité CONTRAT est un triplet: <personne, batiment, Assurance>.



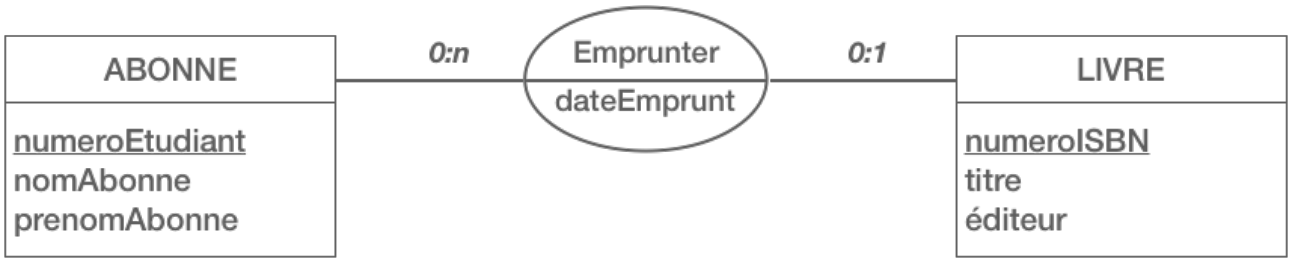
Les relations réflexives



Si l'association/relation lie deux (ou plusieurs) entités du même type, elle est dite "cyclique". Une occurrence 'marié à ' est un couple: < 1 personne/MARI, 1 personne/FEMME >. Dans ce cas, il est primordial de spécifier le rôle de chaque entité afin d'éviter toute ambiguïté.

2.3. Cardinalités

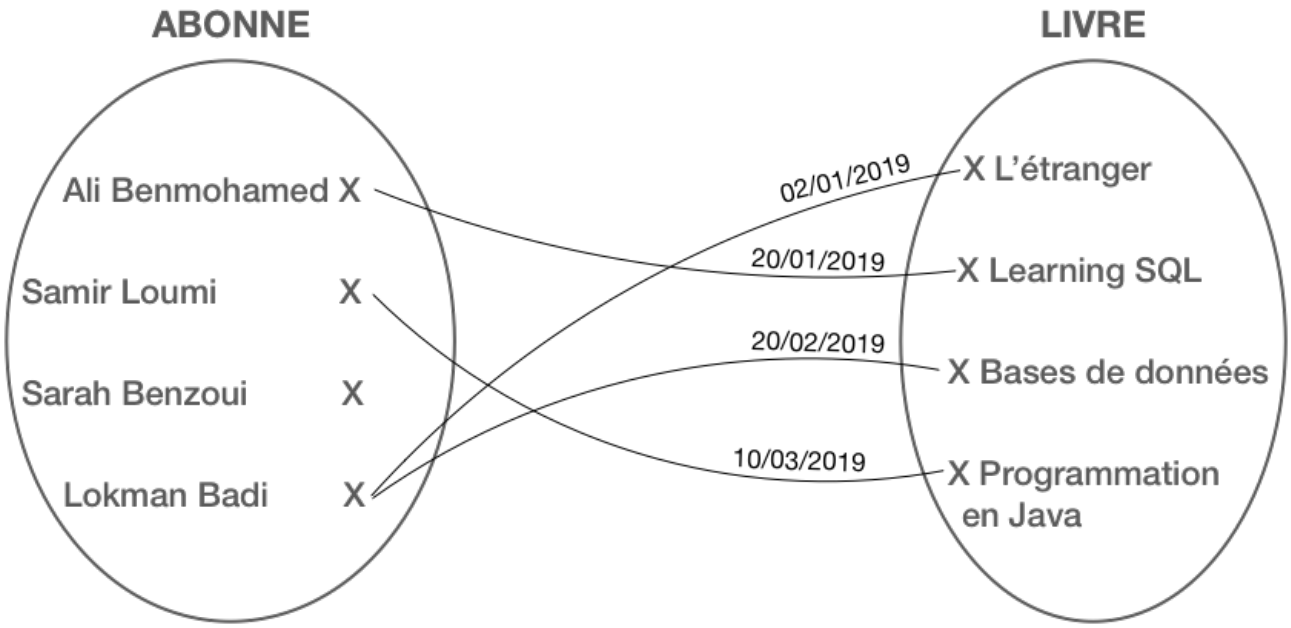
Les cardinalités des rôles permettent de contraindre les associations par les nombres minimum et maximum de participation de chaque entité à l'association. La cardinalité minimale doit être inférieure ou égale à la maximale.



Les cardinalités peuvent être notées comme sur le schéma conceptuel ci-dessus min:max. Il faut lire la cardinalité comme suit :

- Un abonné peut Emprunter plusieurs livres.
- Un livre ne peut être Emprunté que par un seul abonné.

Traduction en modèle informel (ou une instance possible du modèle ci-dessus):



Règles :

- L'expression de la cardinalité est obligatoire pour chaque lien entité-association.
- Il ne peut y avoir de cardinalité maximale égale à 0, car elle rendrait le type-association inutile.
- Une cardinalité minimale est toujours 0 ou 1 et une cardinalité maximale est toujours 1 ou n.
- Si une cardinalité est connue et vaut 2 ou plus, alors nous considérons qu'elle est indéterminée et vaut n. En effet, si cette valeur est définie lors de la conception, il se peut qu'elle évolue dans le futur. Il est donc considérer n comme inconnue dès la conception.

Les seules cardinalités admises sont :

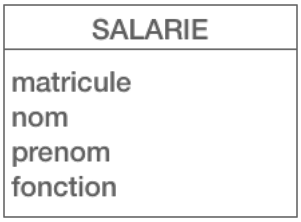
- 0:1 : une occurrence du type-entité peut exister en étant impliquée soit dans aucune association soit au maximum dans une seule

- 0:n : une occurrence du type-entité peut exister en étant impliquée soit dans aucune association soit dans plusieurs associations (sans limite).
- 1:1 : une occurrence du type-entité ne peut exister que si elle est impliquée dans exactement une association.
- 1:n : une occurrence de type-entité ne peut exister que si elle es impliquée dans au moins une association.

2.4. Attributs et identifiants

➡ **Attributs**

Est représentée par un attribut toute information intéressante qui participe à la description d'un objet ou d'un lien et qui ne fait l'objet de traitement qu'en tant que partie de cet objet ou lien. Un attribut ne dépend que de l'entité (ou de l'association, i.e. des entités liées) à laquelle il est attaché.



➡ **Identifiants**

L'identifiant d'une entité ou d'une relation est l'ensemble minimum d'attributs tel qu'il n'existe pas deux occurrences du E (ou A) ayant la même valeur pour ces attributs. Une entité, ou association, peut avoir plusieurs identifiants possibles. Dans certains cas, on ajoute un attribut particulier tel qu'un numéro incrémental, attribut artificiel qui jouera le rôle d'identifiant. Par exemple: n°employé et nom+prénoms sont 2 identifiants possibles de l'entité Employé. Si dans cette entreprise il n'y a jamais 2 employés ayant les mêmes nom et prénoms, ou le même numéro.

3. Le modèle relationnel

Le modèle relationnel à été introduit par CODD chez IBM en 1970. Dans ce modèle, les données sont représentées par des tables. Les tables constituent donc la *structure logique* du modèle relationnel. Au niveau physique, le système est libre d'utiliser n'importe quelle technique de stockage (fichiers séquentiels, indexage, adressage dispersé, séries de pointeurs, compression…)

dès lors qu'il est possible de relier ces structures à des tables au niveau logique. Les tables ne représentent donc qu'une abstraction de l'enregistrement physique des données en mémoire.

Les objectifs du modèle relationnel sont :

- Proposer des schémas de données faciles à utiliser,
- Améliorer l'indépendance logique et physique,
- Mettre à la disposition des utilisateurs des langages de haut niveau ;
- Optimiser les accès à la base de données ;
- Améliorer l'intégrité et la confidentialité ;
- Fournir une approche méthodologique dans la construction des schémas.

De façon très simple, on peut définir le modèle relationnel de la manière suivante :

- Les données sont organisées sous forme de tables à deux dimensions, encore appelées relations, dont les lignes sont appelées n-uplet ou *tuple* en anglais ;
- Les données sont manipulées par des opérateurs de l'algèbre relationnelle ;
- L'état cohérent de la base est défini par un ensemble de contraintes d'intégrité.

Dans la section suivante on va aborder ces terminologies liées et leurs rôles dans le modèle relationnel.

4. Terminologies du modèle relationnel

➡ ***Domaine***

Le domaine d'un attribut est l'ensemble, fini ou infini, de ses valeurs possibles. Par exemple, l'attribut numéro d'étudiant a pour domaine l'ensemble des combinaisons de douze chiffres et le nom et prénom ont pour domaine l'ensemble des combinaisons de lettres.

➡ **Relation**

Une relation est un sous-ensemble du produit cartésien de n domaines d'attributs (n > 0). Une relation est représentée sous la forme d'un tableau à deux dimensions dans lequel les *n* attributs correspondent aux titres des *n* colonnes.

➡ **Attribut**

Un attribut est un identificateur (un nom) décrivant une information stockée dans une base. Exemples d'attribut : l'âge d'une personne, le nom d'une personne, le numéro de sécurité sociale.

➡ **Tuples**

Une occurrence, ou n-uplets, ou tuples, est un élément de l'ensemble figuré par une relation. Autrement dit, une occurrence est une ligne du tableau qui représente la relation.

➡ **Schéma relationnel.**

Un schéma relationnel précise le nom de la relation ainsi que la liste des attributs avec leurs domaines.

5. Règles d'intégrité

Une contrainte d'intégrité est une règle qui définit la cohérence d'une donnée ou d'un ensemble de données de la BD.

Il existe deux types de contraintes :

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

Les contraintes sont définies au moment de la création des tables pour assurer la cohérence logique de la Base de Données.

5.1. Unicité de la clé

Une table contient généralement une colonne ou une combinaison de colonnes dont les valeurs identifient de façon unique chaque ligne dans la table. Cette colonne (ou ces colonnes), appelée clé primaire (PK, Primary Key), assure l'intégrité de l'entité de la table. Les contraintes de clé primaire garantissent des données uniques, c'est pourquoi elles sont souvent définies pour une colonne d'identité.

5.2. Clé étrangère

On appelle « clé étrangère » une colonne ou une combinaison de colonnes utilisée pour établir et conserver une liaison entre les données de deux tables pour contrôler les données qui peuvent être stockées dans la table de clés étrangères. Dans une référence de clé étrangère, la création d'une liaison entre deux tables s'effectue lors du référencement de la ou des colonnes contenant les valeurs de clé primaire d'une table dans la ou les colonnes de l'autre table. Cette colonne devient alors une clé étrangère dans la seconde table.

5.3. Valeurs non nulles

Une telle contrainte spécifie que la valeur d'un attribut doit être renseignée. Par exemple, le degré d'un vin ne pourra être nul, et devra donc être documenté lors de l'insertion d'un vin dans la base, ou après toute mise à jour.

5.4. Contraintes d'intégrité du domaine

Toute valeur prise par un attribut dans le modèle relationnel (ou champ dans la base de données) doit appartenir à un domaine défini pour cet attribut. Cela revient à dire que chaque attribut doit prendre une valeur dans le domaine de valeurs. Par exemple, un jour sera choisi parmi {Dimanche, Lundi, Mardi, …} ; une quantité sera comprise entre 0 et 100.

6. Passage d'un MCD à un modèle relationnel

Un modèle relationnel est composé de relations, encore appelée tables. Ces tables sont décrites par des attributs ou champs (noms de colonnes). Pour décrire une relation, on indique tout simplement son nom en majuscule, suivi du nom de ses attributs entre parenthèses. L'identifiant d'une relation est composé d'un ou plusieurs attributs qui forment la clé primaire. Une relation peut faire référence à une autre en utilisant une clé étrangère, qui correspond à la clé primaire de la relation référencée.

Il n'y a pas de notation officielle pour repérer les clés primaires et étrangères. C'est à vous d'en adopter une et de l'expliquer en légende.
Toutefois, une notation s'est peu à peu répandue dans la modélisation des bases de données et c'est cette notation qu'on va utiliser dans ce cours:

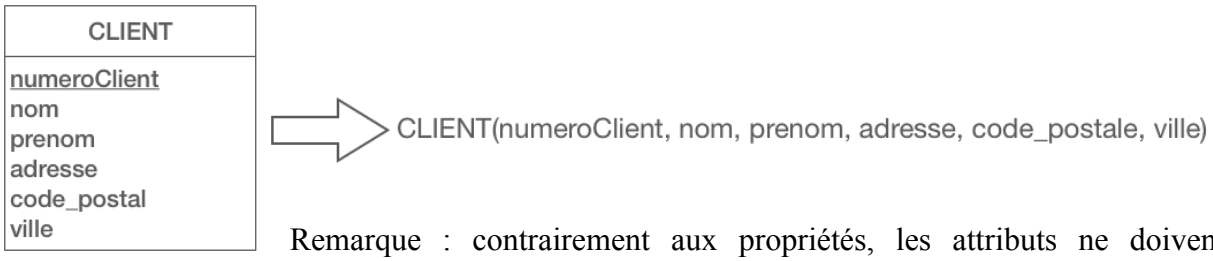
- on souligne la clé primaire d'un seul trait
- on fait précéder (ou suivre) les clés étrangères du symbole #

Chaque ligne (tuple ou enregistrement) d'une table représente une occurrence de l'entité ou de l'association correspondante.

Pour réussite la transformation d'un modèle conceptuel (entité association dans notre cas) vers un modèle relationnel, il faut suivre les étapes suivantes :

➡ **Règle 1 :**

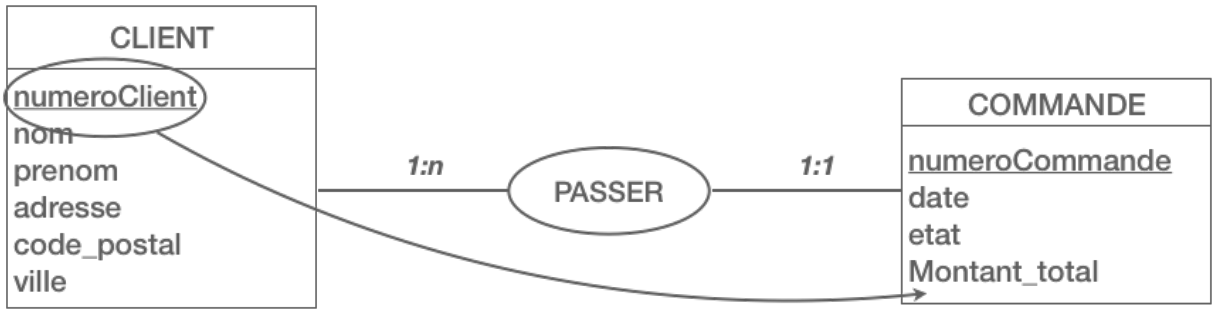
Toute entité devient une relation ayant pour clé primaire son identifiant. Chaque propriété se transforme en attribut.



Remarque : contrairement aux propriétés, les attributs ne doivent pas comporter d'espaces.

➡ **Règle 2 :**

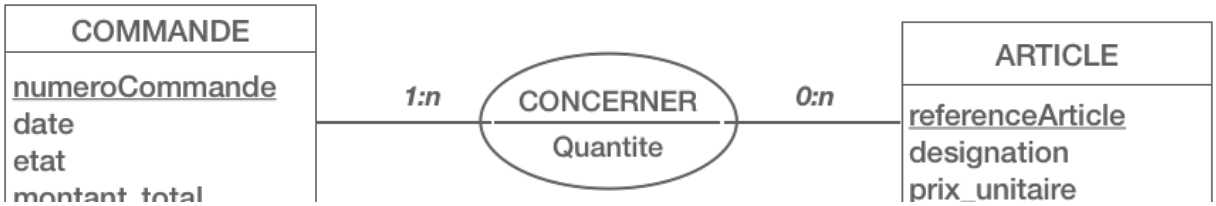
Toute association hiérarchique (de type [1, n]) se traduit par une clé étrangère. La clé primaire correspondant à l'entité père (côté n) migre comme clé étrangère dans la relation correspondant à l'entité fils (côté 1).



COMMANDE(numeroCommande, date, etat, montant_total, #numeroClient)

➡ **Règle 3 :**

Toute association non hiérarchique (de type [n, n] ou de dimension > 2) devient une relation. La clé primaire est formée par la concaténation (juxtaposition) l'ensemble des identifiants des entités



reliées. Toutes les propriétés éventuelles deviennent des attributs qui ne peuvent pas faire partie de la clé.

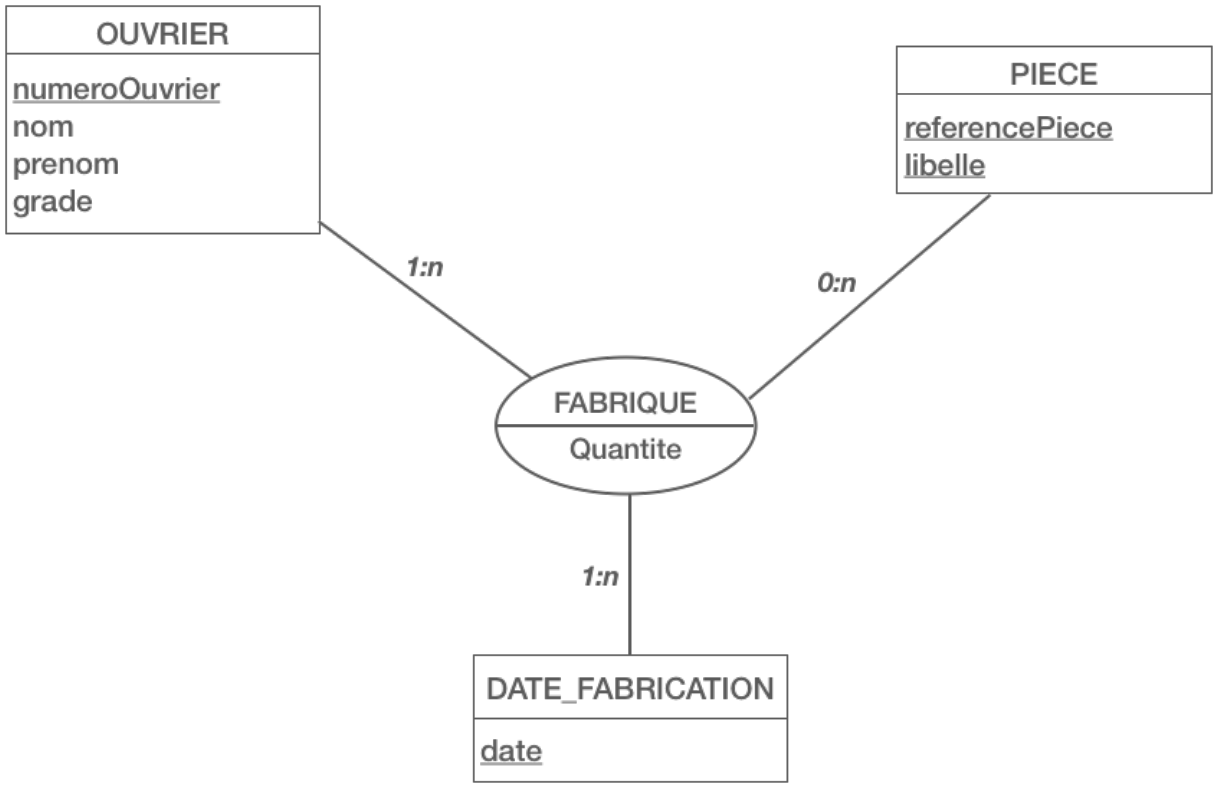
CONCERNER(#numéro_commande, #référence_article, quantité)

Cette règle est valable pour toutes les associations ternaires (ou quaternaires) qui sont forcément non hiérarchiques (cardinalités maximales toutes égales à n).

➡ **Exception à la règle 1**

Les entités n'ayant que leur identifiant comme attribut ne deviennent pas des relations, mais des attributs dans les autres relations liées.

Avec ce modèle, on mémorise chaque jour pour chaque ouvrier les pièces qu'il a fabriqué et en quelle quantité.



Quand on passe au modèle relationnel, l'entité DATE FABRICATION ne devient pas une relation, mais un attribut clé dans la relation FABRIQUE issue de l'association.

~~DATE_FABRICATION(date)~~

FABRIQUE(#numeroOuvrier, #referencePiece, date, quantité).

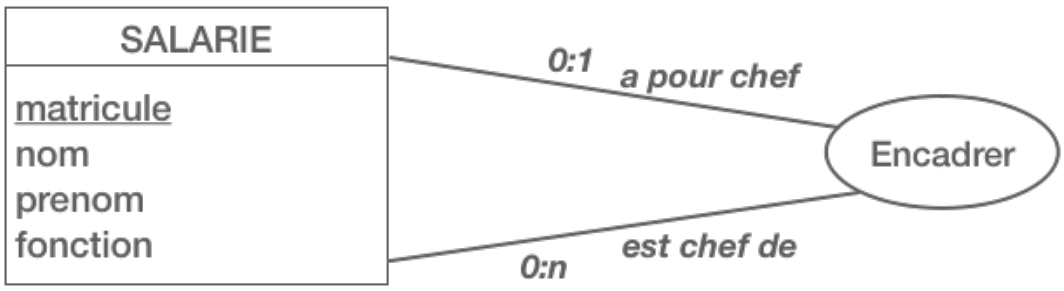
Il est important de mentionner que l'attribut **date** ici fait partie de la clé primaire, mais ce n'est pas clé étrangère.

➡ **Cas particulier des relations réflexives :**

Les associations réflexives suivent les règles 2 ou 3 selon les cardinalités mais posent un problème particulier : une même propriété va se retrouver deux fois en attribut dans la même relation. Il faut alors donner un nom différent et significatif aux deux attributs correspondants.

Dans les réflexives, il est conseillé de nommer les branches par des rôles pour pouvoir lire dans le bon sens l'association. Les rôles aident à nommer les attributs correspondant à l'association.

➡ **Réflexive hiérarchique (une branche à la cardinalité maxi à 1 et l'autre à n)**



Lecture de la relation :
Un salarié a pour chef 0 ou un seul autre salarié. Un salarié est chef de 0 à n autre(s) salarié.

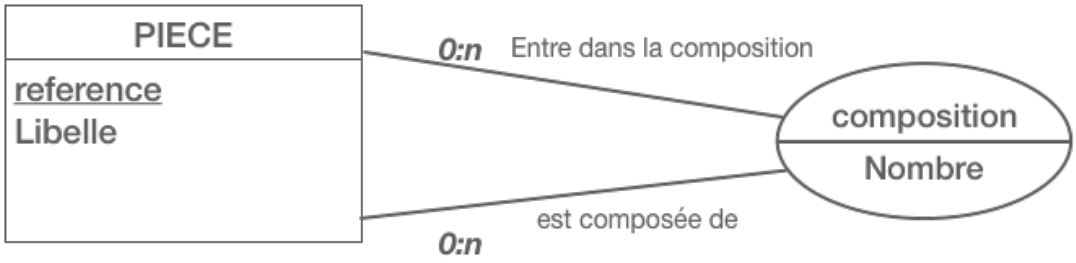
- **Règle n°1:** l'identifiant de SALARIE va devenir clé primaire et les autres propriétés des attributs
- **Règle n°2 :** pour traduire l'association [1, n] encadrer, l'identifiant de l'entité SALARIE devient clé étrangère

L'identifiant de SALARIE matricule se retrouve deux fois dans la relation comme clé primaire et comme clé étrangère. On va donc donner un nom différent et significatif à ces deux matricules, par exemple :

SALARIE(matricule, nom, prenom, fonction, #matricule_chef)

➡ **Réflexive non hiérarchique**

Règle n°3



Une pièce entre dans la composition de 0 à plusieurs autres pièces. Une pièce peut être composée de plusieurs autres pièces. Une pièce entre dans la composition d'une autre un certain nombre de fois.

ex : La pièce "voiture" est composée de 4 pièces "roue". La pièce "roue" est elle-même composée d'une pièce "pneu" et d'une pièce "jante".

Une pièce entrant dans la composition d'une autre est appelée composant. Une pièce composée d'autres pièces est appelée composé. Une roue est à la fois un composant (de voiture) et un composé (de pneu et jante). On obtient les tables suivantes :

PIECE(référence, libellé)

COMPOSITION(#référence_composé, #référence_composant, nombre)

7. Exercices

Exercice 1

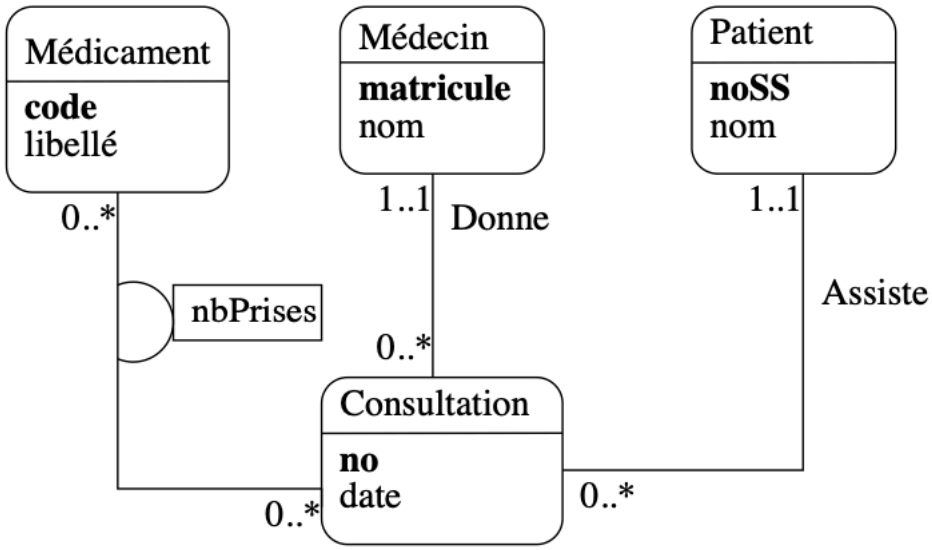
On vous donne un schéma E/A ci-après représentant des visites dans un centre médical. Répondez aux questions suivantes en fonction des caractéristiques de ce schéma.

1. Un patient peut-il effectuer plusieurs visites ?

2. Un médecin peut-il recevoir plusieurs patients dans la même consultation ?

3. Peut-on prescrire plusieurs médicaments dans une même consultation ?

4. Deux médecins différents peuvent-ils prescrire le même médicament ?



Exercice 2

On trouve dans un SGBD relationnel les relations ci-dessous. Les clés primaires sont soulignées et les clés étrangères ne sont pas signalées.

– Immeuble (nom, adresse, nbEtages, annéeConstruction, nomGérant)

– Appart (nomImm, noApp, type, superficie, étage)

– Personne (nom, prenom, age, codeProfession)

– Occupant (nomImm, noApp, nomOccupant, annéeArrivée)

– Propriété (nomImm, nomPropriétaire, quotePart)

– TypeAppart (code, libellé)

– Profession (code, libellé)

Question : identifier les clés étrangères dans chaque relation, et reconstruire le schéma E/A.

8. Language SQL

Le **SQL** (Structured Query Language) est un langage permettant de communiquer avec une base de données. Ce langage informatique est notamment très utilisé par les développeurs web pour communiquer avec les données d'un site web.

SQL est un langage déclaratif, il n'est donc pas a proprement parlé un langage de programmation, mais plutôt une interface standard pour accéder aux bases de données.

Dans ce support de cours des explications sur les principales commandes pour lire, insérer, modifier et supprimer des données dans une base sont présentées.

8.1. Création d'une base de données

Pour créer une base de donnée, il faut écrire la commande suivante :

Remplacez <db_name> par le nom de votre base de données sans espace aucun.

```
1 CREATE DATABASE db_name;
```

8.2. Création des tables

La création de table est le fondement de la création d'une base de données en SQL. La syntaxe pour créer n'importe quelle table est la suivante :

```
1 CREATE TABLE nom_table (
2  nom_colonne1 domaine1,
3  nom_colonne2 domaine2,
4  ...
5  nom_colonneN domaineN
6 );
```

Exemple réel :

```
1  CREATE TABLE Personne (
2  Nom VARCHAR(25),
3  Prenom VARCHAR(25),
4  Age NUMERIC(3)
5 );
```

Domaines de données

Un attribut d'une relation est défini pour un certain domaine ou type. 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 standard sont :

INTEGER OU INT, SMALLINT, FLOAT, CHAR(X), VARCHAR(X), DATE, DATETIME, …

8.3. 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 base de données.

```
1 CREATE TABLE nom_table (
2  nom_colonne1 domaine1 <contraintes colonne1>,
3  nom_colonne2 domaine2 <contraintes colonne2>,
4  ...
5  nom_colonneN domaineN <contraintes colonneN>,
6 <contraintes de table>
7 );
```

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

Syntaxe

```
1 CREATE TABLE Personne (
2  N°SS CHAR(13) PRIMARY KEY,
3  Nom VARCHAR(25) NOT NULL,
4  Prenom VARCHAR(25) NOT NULL,
5  Age INTEGER(3) CHECK (Age BETWEEN 18 AND 65),
6  Mariage CHAR(13) REFERENCES Personne(N°SS),
7  UNIQUE (Nom, Prenom)
8 );
```

Exemple de contraintes d'intégrité

```
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.

8.4. Manipulation de données : Ajout, Modification, Suppression

Une fois vous avez créé vos tables obtenues après le passage d'un modèle entité-association au modèle relationnel, vous pouvez maintenant peupler la base. Vous pouvez insérer, modifier et supprimer des données.

➡ **Ajouter**

La syntaxe pour ajouter des données est la suivante :

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

Exemple :

```
1 INSERT INTO Personne (Nom, Prenom, Age)
2 VALUES ('Benamar','Ali', 21),
```

➡ **Mise à jour de données**

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

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

Exemple :

```
1 UPDATE Personne
2 SET Prenom='Mohamed Ali', Age=22
3 WHERE Nom = 'Benamar'
```

➡ **Suppression de données**

La syntaxe pour ajouter des données est la suivante :

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

Exemple :

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

8.5. Interrogation de données

La commande SELECT constitue la commande permettant d'interroger une base de données. Elle permet de :

- sélectionner certaines colonnes d'une table (projection) ;
- sélectionner certaines lignes d'une table en fonction de leur contenu (sélection) ;
- combiner des informations venant de plusieurs tables (jointure, union, intersection, différence et division) ;
- combiner entre elles ces différentes opérations.

Une requête (*i.e.* une interrogation) est une combinaison d'opérations portant sur des tables (relations) et dont le résultat est lui-même une table dont l'existence est irréelle (le temps de la requête).

Une requête se présente généralement sous la forme :

SELECT [ALL|DISTINCT] chaine de sélection FROM nom de table [SYNONYME] [WHERE condition]

Paramètres :

- La chaine de sélection est la liste des colonnes sur lesquelles on fait une projection.

- La clause FROM précise la table sur laquelle la recherche est faite.

- La clause WHERE précise une condition pour critère de s´election.

Des exemples d'utilisation de la commande SELECT :

Obtenir le nom et le prénom de tous les patients

```
SELECT nom, prenom FROM Personne
```

Obtenir l'ensemble des informations relatives aux traitements

```
SELECT *
FROM Traitement
```

Lister les noms des personnes de plus de 18 ans

```
SELECT nom FROM Patient WHERE age>18
```

Obtenir les traitements dont la posologie est '1 fois par jour' ou '2 fois par jour'

```
SELECT *
FROM Traitement
WHERE posologie='1 fois par jour' OR posologie='2 fois par jour'
```

Obtenir le nom des patients ayant entre 20 et 30 ans et commençant par 'A'

```
SELECT nom
FROM Patient
WHERE age BETWEEN 20 AND 30 AND nom LIKE 'A%'
```

8.6. Les fonctions de calcul : MIN, MAX, COUNT, SUM, AVG

Les fonctions de calcul, appelées aussi les fonctions d'agrégation, sont utilisées pour effectuer les calculs sur plusieurs lignes d'une seule colonne d'une table. Elle retourne une valeur unique. Elles sont également utilisées pour résumer les données.

Remarque ! Toutes les fonctions d'agrégation excluent par défaut les valeurs **NULL** avant de travailler sur les données.

➡ **COUNT**

La fonction **COUNT** est utilisée pour compter le nombre de lignes dans une table de base de données. Il peut fonctionner sur les types de données numériques et non numériques.

La fonction **COUNT** utilise **COUNT(*)** qui renvoie le nombre de toutes les lignes d'une table spécifiée. **COUNT(*)** considère les doublons et Null.

Exemples d'utilisation :

```
SELECT count(DISTINCT Age) FROM Employes
```

SELECT count(numEtudiant) **FROM** Etudiants

➡ **SUM**

La fonction **SUM** renvoie la somme de toutes les valeurs de la colonne spécifiée. **SUM** fonctionne uniquement sur les champs numériques.

Exemple :

```
SELECT SUM(Salaire) FROM Employés
```

➡ **AVG**

La fonction **AVG** renvoie la moyenne des valeurs d'une colonne spécifiée. Tout comme la fonction **SUM**, elle ne fonctionne que sur les types de données numériques.

Exemple :

SELECT AVG(Salaire) **FROM** Employes

➡ **MIN**

La fonction **MIN** est utilisée pour déterminer la plus petite valeur de toutes les valeurs sélectionnées d'une colonne.

Exemple :

SELECT MIN(Salaire) **FROM** Employes;

➡ **MAX**

Comme son nom l'indique, la fonction **MAX** est l'opposé de la fonction **MIN**. Elle renvoie la plus grande valeur de toutes les valeurs sélectionnées d'une colonne.

Exemple :

```
SELECT MAX(Salaire) FROM Employes;
```

8.7. Les opérateurs de comparaison

Les opérateurs de comparaison SQL sont utilisés pour comparer deux arguments et renvoi vrai ou faux en accord avec la condition spécifiée dans l'instruction. Le tableau suivant résume ces opérateurs.

8.8. Projection

La projection est réalisée par la clause SELECT, le produit cartésien par la clause FROM et la sélection par la clause WHERE.

8.9. Tri et groupement de lignes

Dans une requête de sélection, on peut obtenir le résultat trié et/ou groupé.

Opérateur	Description	Exemple
=	Retourne la valeur vrai si l'argument de gauche équivaut à l'argument de droite.	a = b
>	Retourne la valeur vrai si l'argument de gauche possède une valeur supérieure à celle de l'argument de droite.	a > b
<	Retourne la valeur vrai si l'argument de gauche possède une valeur inférieure à celle de l'argument de droite.	a < b
>=	Retourne la valeur vrai si l'argument de gauche possède une valeur supérieure ou égale à celle de l'argument de droite	a >= b
<=	Retourne la valeur vrai si l'argument de gauche possède une valeur inférieure ou égale à celle de l'argument de droite	a <= b
<>	Retourne la valeur vrai si l'argument de gauche n'équivaut pas à l'argument de droite	a <> b
!>	Retourne la valeur vrai si l'argument de gauche possède une valeur qui n'est pas supérieure ou égale à celle de l'argument de droite	a >! b
<!	Retourne la valeur vrai si l'argument de gauche possède une valeur qui n'est pas inférieure ou égale à celle de l'argument de droite	a <! b

```
SELECT [ALL|DISTINCT] chaine de sélection
FROM liste de tables
[WHERE condition]
[ORDER BY liste de colonnes [ASC|DESC]] (mise en forme) ou

[GROUP BY liste de colonnes [HAVING condition]] (moyen de trouver
un résultat)
```

Clauses :

ORDER BY : tri par ordre croissant ou décroissant.

GROUP BY : réarrange la table en groupes de lignes `a raison d'un groupe pour chaque valeur différente associée à la liste de colonnes.

HAVING : permet de spécifier une condition qui sera évaluée pour chaque groupe.

8.10. Notion de jointure

Les jointures en SQL permettent d'associer plusieurs tables dans une même requête. Cela permet d'exploiter la puissance des bases de données relationnelles pour obtenir des résultats qui combinent les données de plusieurs tables de manière efficace.

En général, les jointures consistent à associer des lignes de 2 tables en associant l'égalité des valeurs d'une colonne d'une première table par rapport à la valeur d'une colonne d'une seconde table. Imaginons qu'une base de 2 données possède une table "utilisateur" et une autre table "adresse" qui contient les adresses de ces utilisateurs. Avec une jointure, il est possible d'obtenir les données de l'utilisateur et de son adresse en une seule requête.

On peut aussi imaginer qu'un site web possède une table pour les articles (titre, contenu, date de publication …) et une autre pour les rédacteurs (nom, date d'inscription, date de naissance …). Avec une jointure il est possible d'effectuer une seule recherche pour afficher un article et le nom du rédacteur. Cela évite d'avoir à afficher le nom du rédacteur dans la table "article".

Il y a d'autres cas de jointures, incluant des jointures sur la même table ou des jointure d'inégalité. Ces cas étant assez particulier et pas si simple à comprendre, ils ne seront pas élaboré sur cette page.

Exemple d'une jointure :

```
SELECT Nom, Prenom FROM Etudiant, Emprunter WHERE
Etudiant.numEtudiant = Emprunter.numEtudiant
```

Cette requête renvoie la liste des étudiants(nom et prénom) des étudiants qui ont emprunté des livres.

8.11. Exercice sur SQL

Exercice 1

On considère ci-dessous une base de données où sont stockées les informations relatives à des transactions immobilières.

Les biens immobiliers sont identifiés par un numéro, et on connaît leur type (appartement, maison, parking, ...), ainsi que leur nombre de pièces, adresse, ville et département.

Chaque client est identifié par un numéro, et on conserve son nom, prénom, mail et téléphone. L'historique des visites des différents biens immobiliers est également conservé dans la base de données.

Les notaires en charge des achats de biens immobiliers sont identifiés par un numéro, et on stocke leur nom, adresse et ville.

Chaque achat est consigné dans la base de données et identifié par un numéro. Chaque achat est relatif à un bien immobilier, un client et un notaire donnés, et est associé à un prix et à une date d'achat.

Les tables de la base sont les suivantes :

BienImmo (nBien, type, nbPieces, adr, ville)

Client (nCli, nomCli, prénomCli, mail, téléphone)

Visite (#nCli, #nBien, dateVis)

Notaire (nNot, nomNot, adrNot, villeNot)

Achat (nAchat, #nBien, #nCli, #nNot, prix, dateAch)

Les clés primaires sont soulignées d'un trait plein, les clés étrangères sont soulignées en pointillés.

Traduire en SQL les requêtes suivantes :

1. Adresse des appartements à Annaba ou à Constantine achetés en 2015.

2. Nom des clients qui n'ont jamais visité de maison à Annaba.

3. Nom des notaires ayant été chargés d'achats de biens immobiliers dans la ville d'Annaba et dans à Constantine.

4. Nom du client et prix d'achat du bien immobilier correspondant à l'achat le plus récent.

5. Mail des clients ayant effectué au moins 3 visites.

6. Prix moyen des appartements de 2 pièces dans les villes Annaba, Constantine et Guelma.

7. Nombre de visites reçues par chaque bien immobilier -seulement pour les biens ayant reçu au minimum 10 visites, trié du plus grand au plus petit (en nombre de pièces).

Exercice 2

On suppose qu'une bibliothèque gère une base de données dont le shéma est le suivant (les clés primaires des relations sont soulignées) :

Emprunt(Personne, Livre, DateEmprunt, DateRetourPrevue, DateRetourEffective)

Retard(Personne, Livre, DateEmprunt, PenalitéRetard)

Exprimer les requêtes suivantes en algèbre relationnelle en SQL.

1. Quelles sont les personnes ayant emprunté le livre "Recueil Examens BD" ?

2. Quelles sont les personnes n'ayant jamais rendu de livre en retard ?

3. Quelles sont les personnes ayant emprunté tous les livres (empruntés au moins une fois) ?

4. Quels sont les livres ayant été empruntés par tout le monde (i.e. tous les emprunteurs) ?

5. Quelles sont les personnes ayant toujours rendu en retard les livres qu'elles ont empruntés ?

III Application avec ACCESS

Après avoir vu les étapes de création d'une base de données, nous allons voir dans cette section comment implémenter le modèle logique de données avec un SGBD relationnel. Dans notre cas, on va utiliser ACCESS.

Microsoft Access (officiellement Microsoft Office Access) est une base de données relationnelle éditée par Microsoft. Ce logiciel fait partie de la suite Microsoft Office.

MS Access est composé de plusieurs programmes : le moteur de base de données Microsoft Jet, un éditeur graphique, une interface de type Query by Example pour interroger les bases de données, et le langage de programmation Visual Basic for Applications.

Une base de données Access se compose de Tables, requêtes, formulaires, macros et modules. Un fichier de base de données Access se compose de : Tables. Les tables stockent les données. Elles ne sont basées sur rien d'autre.

1. Requêtes

Les requêtes permettent de faire des filtres, tris et calculs. Elles permettent notamment de synthétiser les données en provenance de plusieurs tables liées.

Une requête peut-être basée sur une table, ou sur une autre requête. On peut ainsi créer une requête basée sur une autre requête, basée sur encore une autre requête, finalement basée sur une table.

Les requêtes permettent également d'effacer ou de modifier les données en masse et automatiquement.

1. Formulaires

Ce sont des masques de saisie. Bien qu'on puisse saisir des données directement dans les tables ou les requêtes, les formulaires permettent une saisie vraiment personnalisée et ultra-optimisée.

1. Etats

Ce sont des aperçus avant impression. Bien qu'on puisse imprimer les formulaires, requêtes et tables, l'état propose une impression réellement plus puissante, plus personnalisée. On ne peut pas

saisir des données dans un état. Un état se base sur une requête ou une table, mais pas sur un formulaire.

Les états ne s'impriment pas forcément sur papier, mais peuvent être juste consultés à l'écran.

1. Macros

Ce sont des instructions qui permettent d'augmenter notablement la personnalisation de la base de données. Une macro peut s'exécuter indépendamment, ou être associée à un état, un formulaire.

Depuis la version 2007, il est également possible d'associer des macros à des tables.

Dans la section ci-après, on va expliquer les étapes de création d’une base de données à partir du modèle relationnel obtenu à la fin du processus de conception d’une base de donnée.

IV Etude de cas avec ACCESS

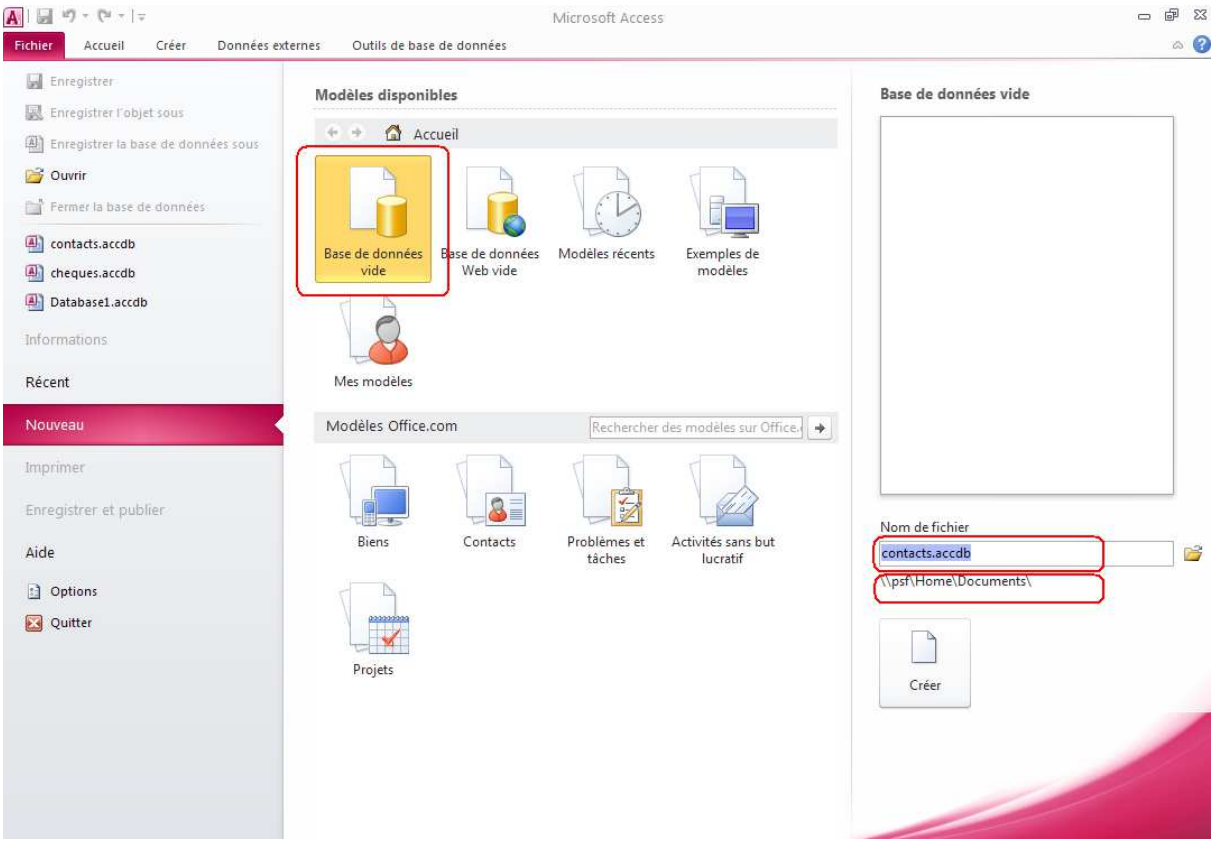
Dans cette section, nous allons présenter un exemple illustratif de création d'une base de données en utilisant ACCESS. L'exemple pris est sur la gestion des contacts commerciaux. Le modèle relationnel des tables est le suivant :

- nature_contact(code, intitulé)
- type_prospect(code, intitulé)
- table_prospect(code, nom, prenom, adresse1, adresse2, ville, cp, tel, code_type_prospect, photo)
- table_prospect(code, #code_prospect, date, heure, contenu, date_prochain_contact, #code_nature_contact)

Etape 1 : Création de la base de données

➡ Base de données vide

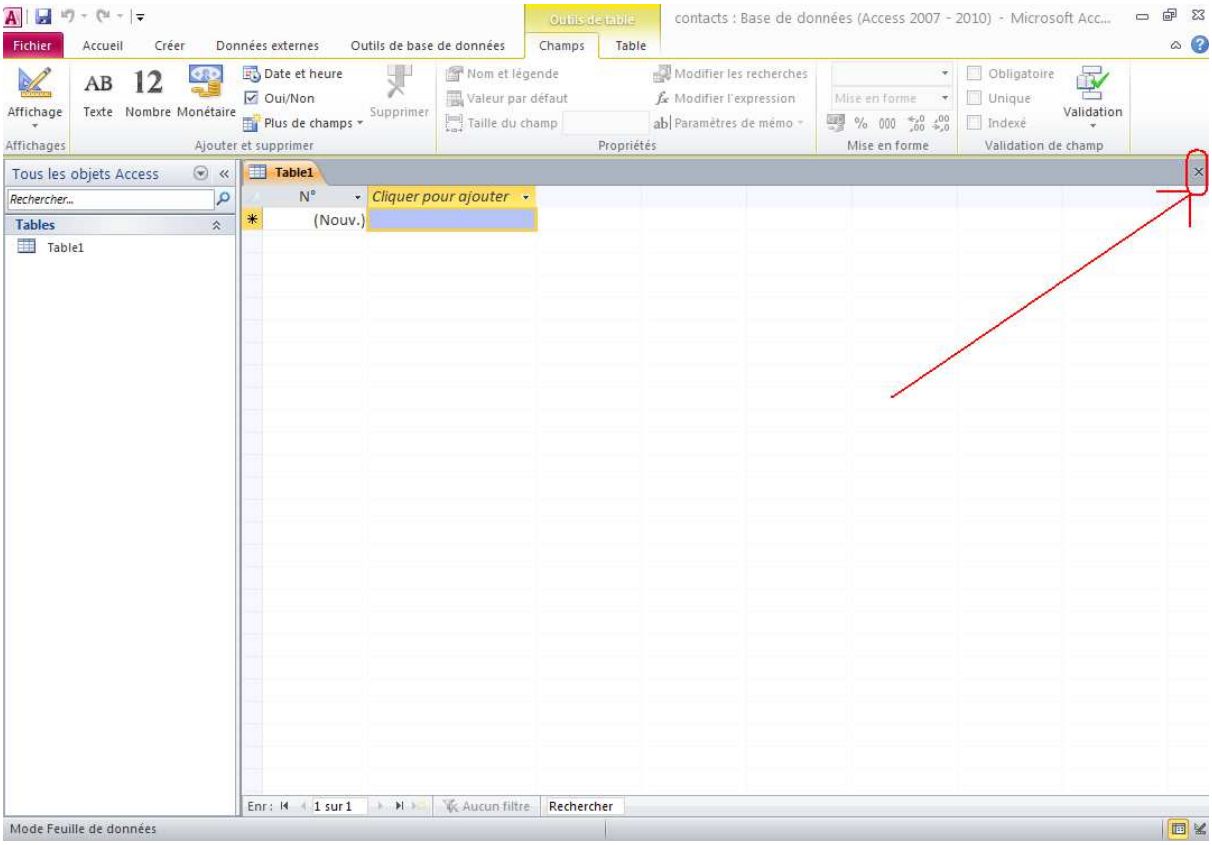
➡ Nom du fichier : contacts.accdb



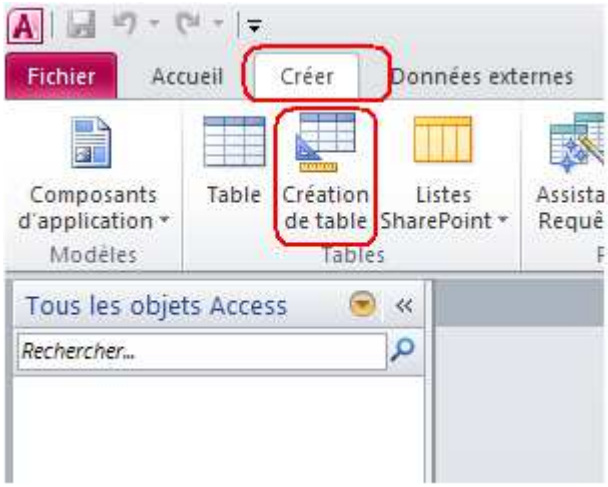
➡ Indiquer l'emplacement

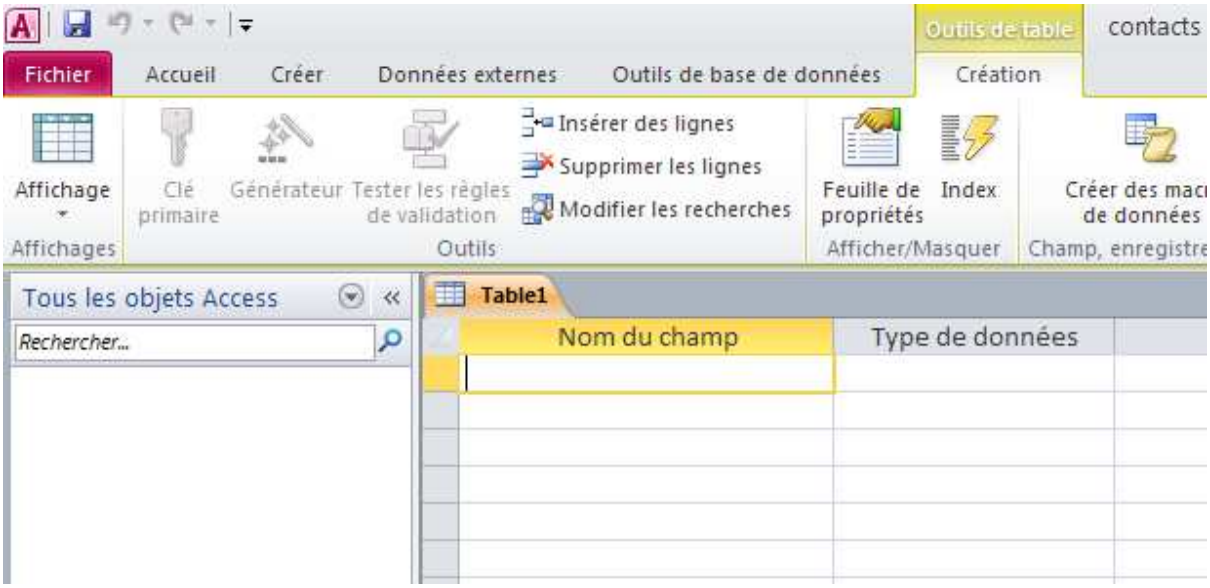
➡ Créer

Une fois créée, on obtient la fenêtre suivante :

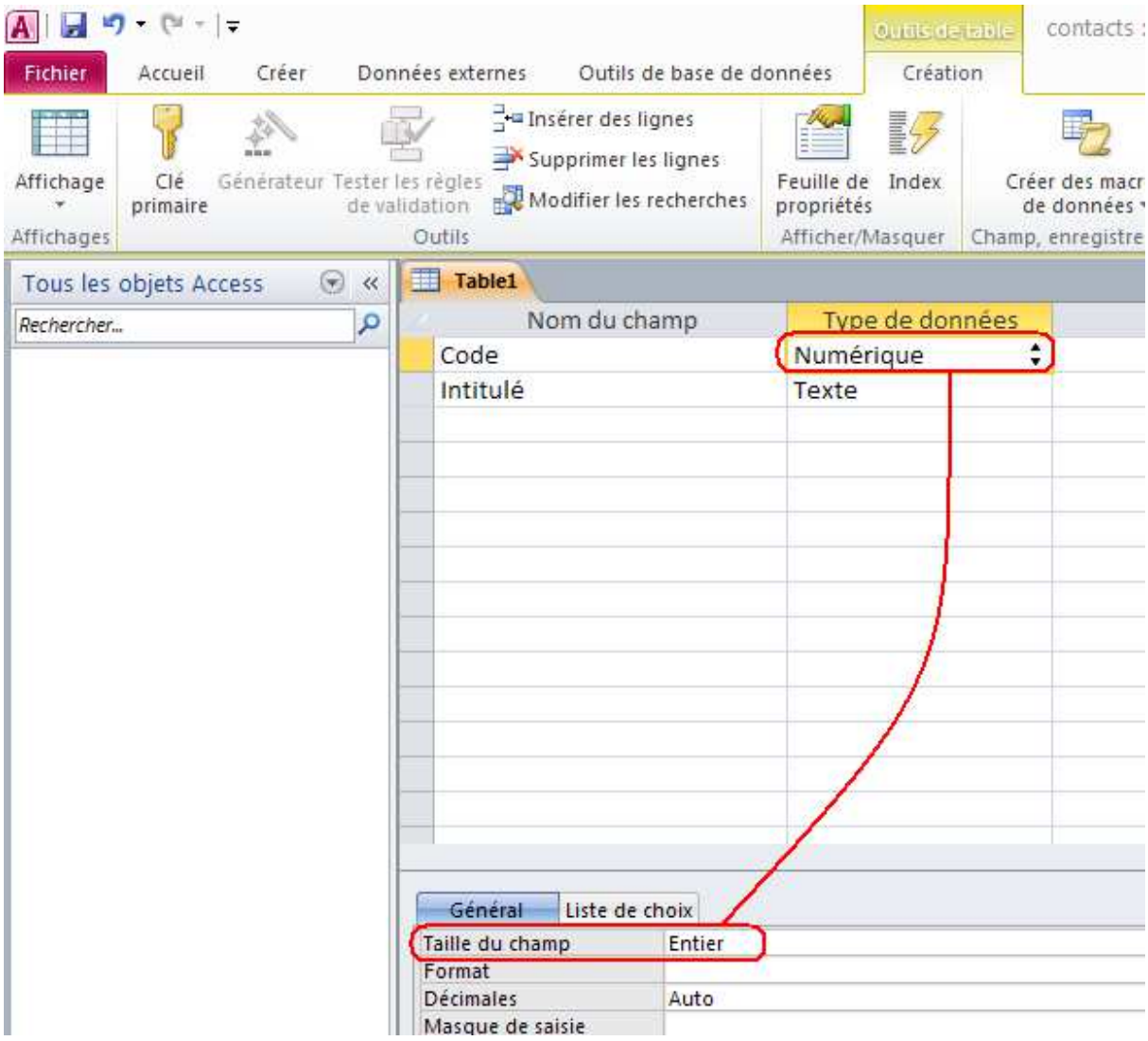


Etape 2 : Création des tables



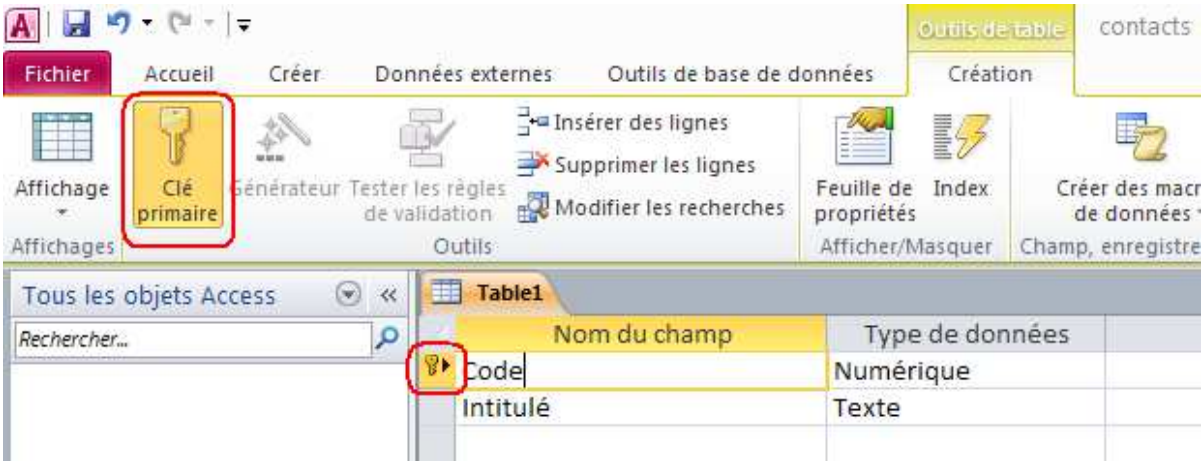


➡ Entrer les 2 champs Code et Intitulé :

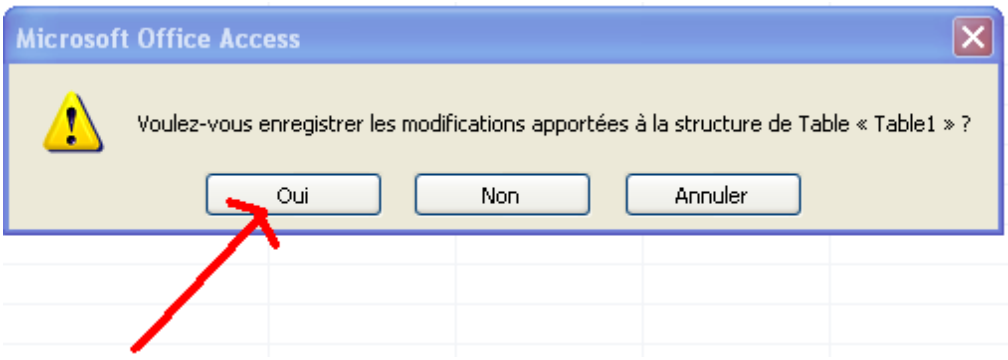


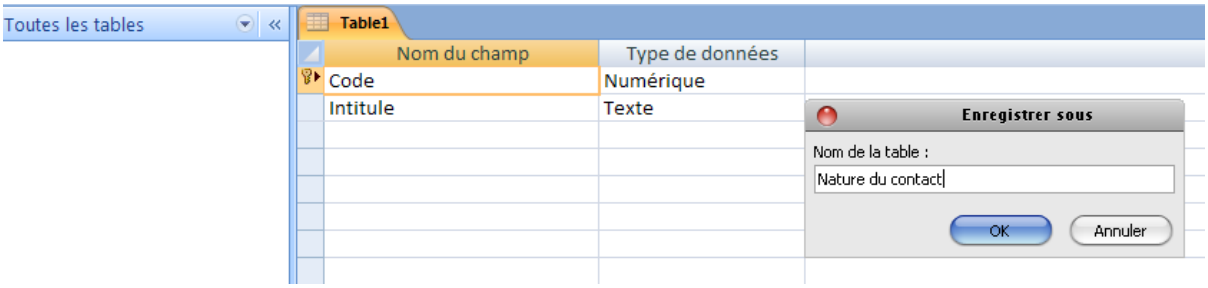
Ne reste plus qu'à indiquer que le code est la clé primaire :

➡ Bouton droit de la souris sur le Code. Sélectionner clé primaire (ou bouton Clé primaire), ce qui donne :

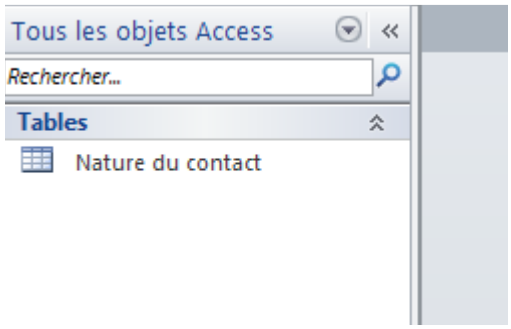


Fermer la fenêtre de création de la table et l'enregistrer :





Une première table est créée :



Créer les 3 autres de la même manière.

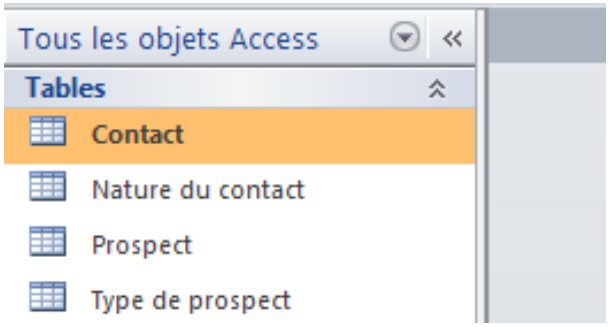
Précisions :

- Afin de modifier la structure de la table : Bouton droit de la souris, Mode création.

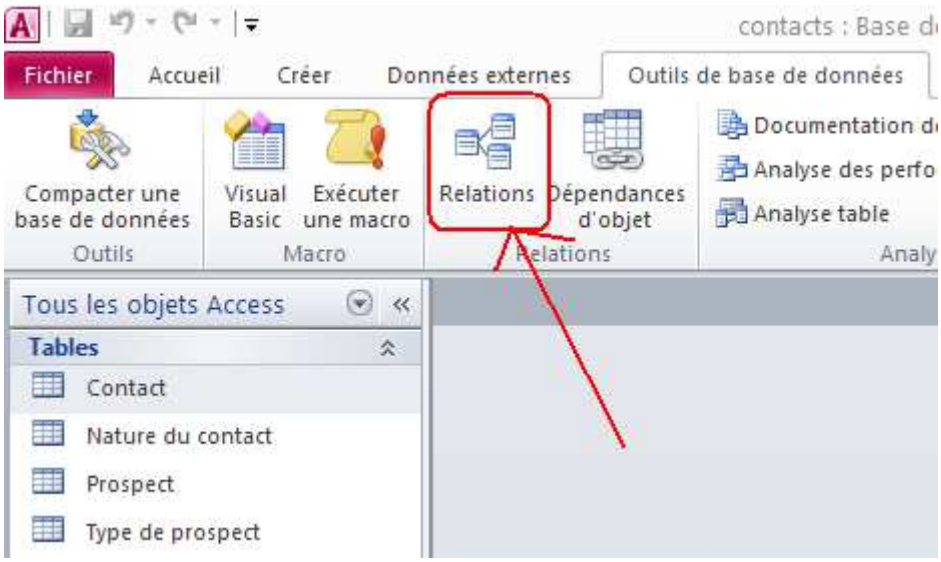
- Afin de visualiser son contenu (vide actuellement = la table nature du contact ne contient aucun enregistrement, double cliquer sur la table.

IMPORTANT : Toujours fermer la fenêtre après chaque étape de création ou modification (a pour effet de les enregistrer).

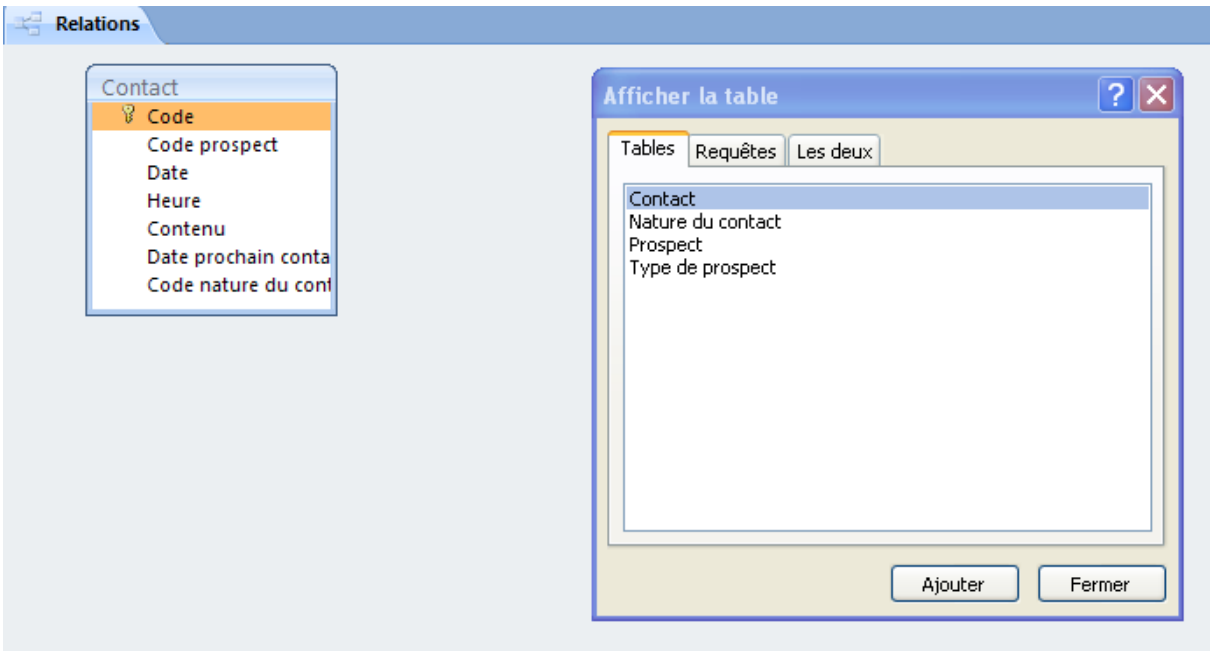
Créer les autres tables de la même manière



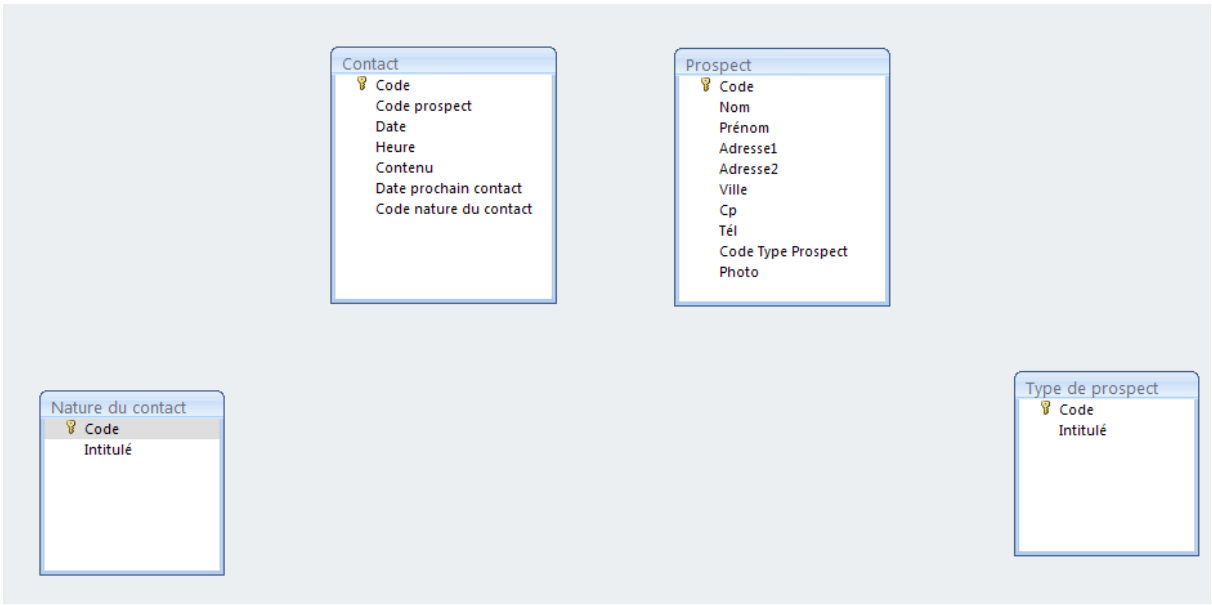
Etape 3 : Relation entre les tables



Ajouter les tables :



Pour obtenir les quatre tables suivantes :



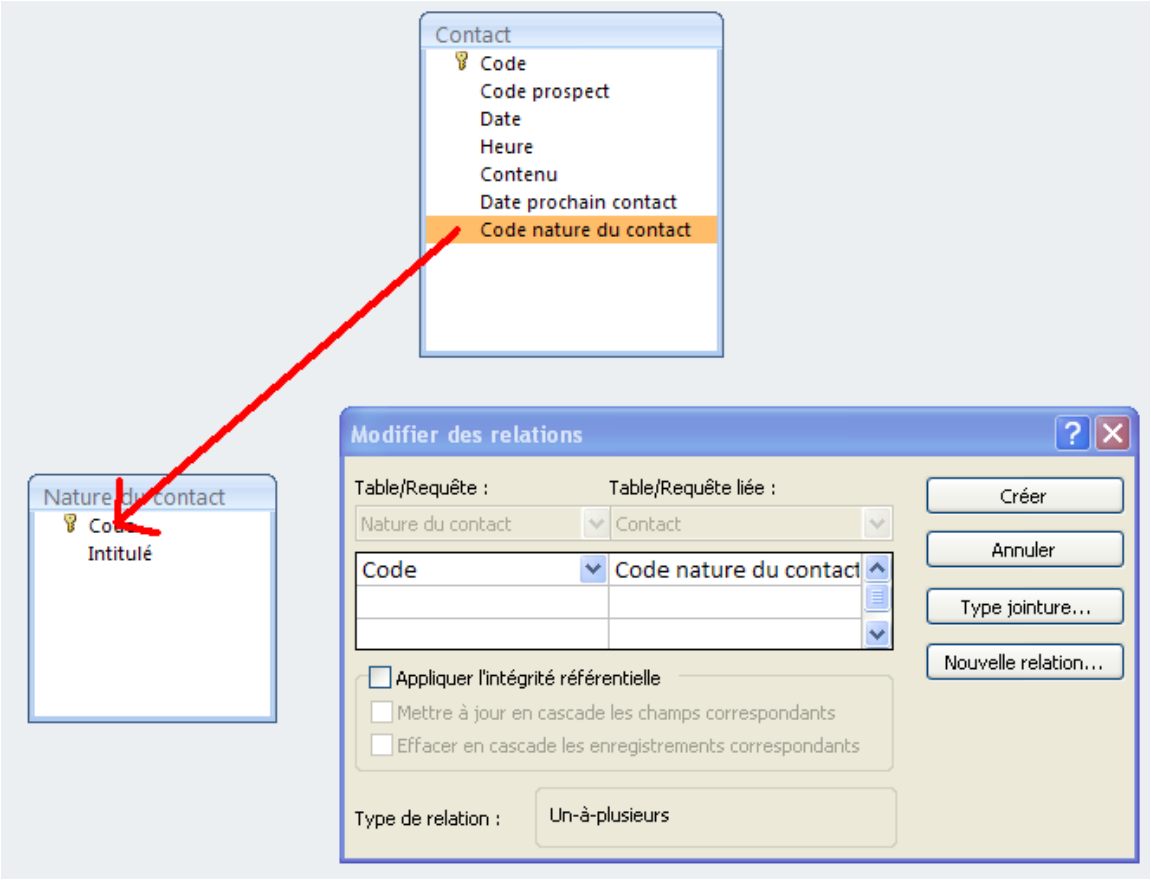
Relation entre 2 tables : Ex : Contact <-> Nature du contact

Faire glisser le champ Code nature du contact (de la table contact) sur le champ Code (de la table Nature du contact)

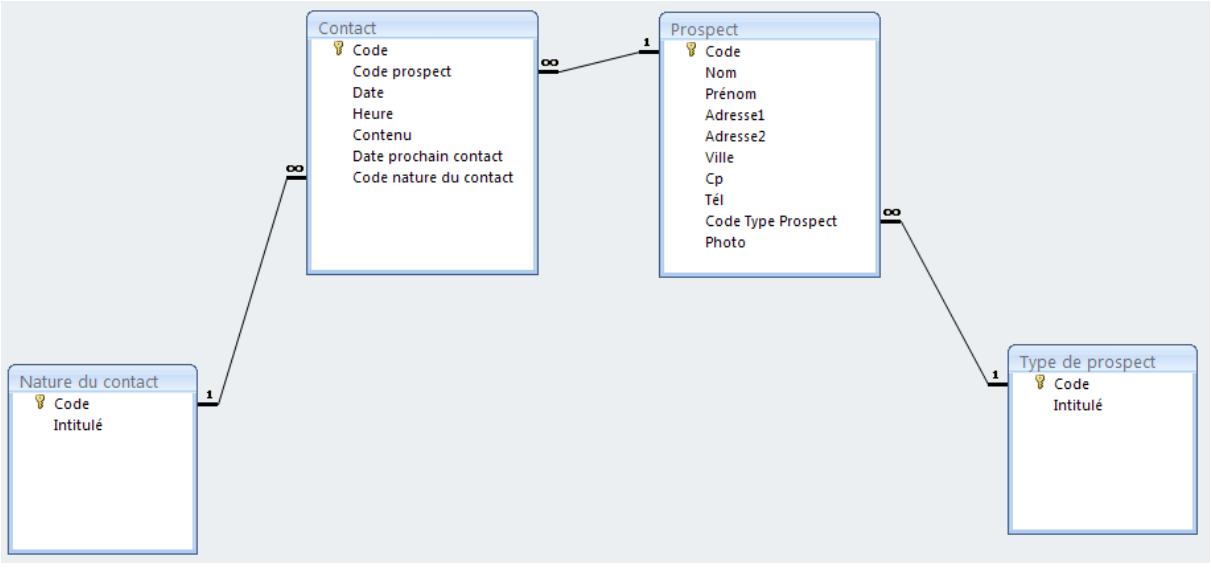
La sous fenêtre qui s'affiche a pour objectif la définition les caractéristiques de la relation, à savoir : les cardinalités, si la suppression d'un enregistrement d'une table implique la suppression des enregistrements de l'autre tables (effacement en cascade), etc.

- A un contact correspond une et une seule nature du contact
- A une nature du contact correspond un ou plusieurs contacts

Traiter les autres relations (en appliquant l'intégrité référentielle) :



Fermer et enregistrer les relations.



Bibliographie

Michelle Clouse, Algèbre relationnelle Guide pratique de conception d'une base de données relationnelle normalisée.Édition : ENI - **377 pages** , 10 mars 2008. ISBN10 : 2746041547 - ISBN13 : 9782746041547

Chris-J Date, Introduction aux bases de données. Vuibert - **1045 pages** , 1er décembre 2004. ISBN10 : 2711748383 - ISBN13 : 9782711748389

Anne-Christine Bisson, SQL - Les fondamentaux du langage. Editions ENI; 3e édition, 409 pages. 13 décembre 2017 ISBN-13 : 978-2409011429 ISBN-10 : 240901142X

Colin Ritchie, *Database Principles and Design,* Cengage Learning EMEA - 2008, ISBN 9781844805402

Philip J. Pratt et Joseph J. Adamski, *Concepts of Database Management*, Cengage Learning - 2011. ISBN 9781111825911

Sikha Bagui et Richard Earp, Database Design Using Entity-Relationship Diagrams, CRC Press - 2011. ISBN 9781439861769

Jean-Luc Hainaut, Bases de données - 4e édition. Concepts, utilisation et développement. octobre 2018