Structure de base de données

Introduction :

Nous savons modéliser une base de données et la traduire en un schéma relationnel exploitable par le système informatique qui aura la charge de sa mise en œuvre. Cependant, avant de concrétiser la création informatique de la structure de cette base de données, il est nécessaire de procéder à un contrôle. L'objectif de cette étape est de repérer d'éventuelles anomalies dans le schéma relationnel qui pourraient être à l'origine de redondances d'informations ou d'incohérences.

Nous allons aborder dans ce cours la notion de dépendance fonctionnelle sur laquelle s'appuie le procédé de normalisation. Nous étudierons ensuite ce procédé, et verrons dans quelle mesure il sert à identifier des anomalies dans le schéma. Pour terminer, nous étudierons les contraintes d'intégrité qui nécessitent d'effectuer des choix sur le comportement de la base de données lors de certaines opérations de mises à jour dans ses tables.

1 Dépendance fonctionnelle

Intéressons-nous donc en premier lieu à la notion de dépendance fonctionnelle.

Définition

Dépendance fonctionnelle :

Soient A et B deux ensembles d'attributs. On dit que B dépend fonctionnellement de A, si, à chaque valeur prise par A (c'est-à-dire chaque occurrence), ne correspond qu'une et une seule valeur de B. On note cette dépendance $A \rightarrow B$.

Astuce

> On peut utiliser l'abrégé « DF » pour le terme « dépendance fonctionnelle ».

Exemple

> À une immatriculation de véhicule ne correspond forcément qu'un modèle de voiture. Il existe donc une DF, entre les attributs « immatriculation » et « modèle ».
> En revanche, lorsque l'on s'intéresse à un modèle de véhicule, on ne lui trouvera pas forcément une seule plaque d'immatriculation... Heureusement pour le constructeur automobile !

Il est possible de caractériser une DP.

→ On observera si une dépendance fonctionnelle est **élémentaire** et si elle est **directe**.

a. Dépendance fonctionnelle élémentaire

Définition

> **DF élémentaire :**
>
> Soient A et B deux ensembles d'attributs tels que $A \rightarrow B$. La dépendance $A \rightarrow B$ est élémentaire s'il n'existe pas un attribut C contenu dans A tel que $C \rightarrow B$.

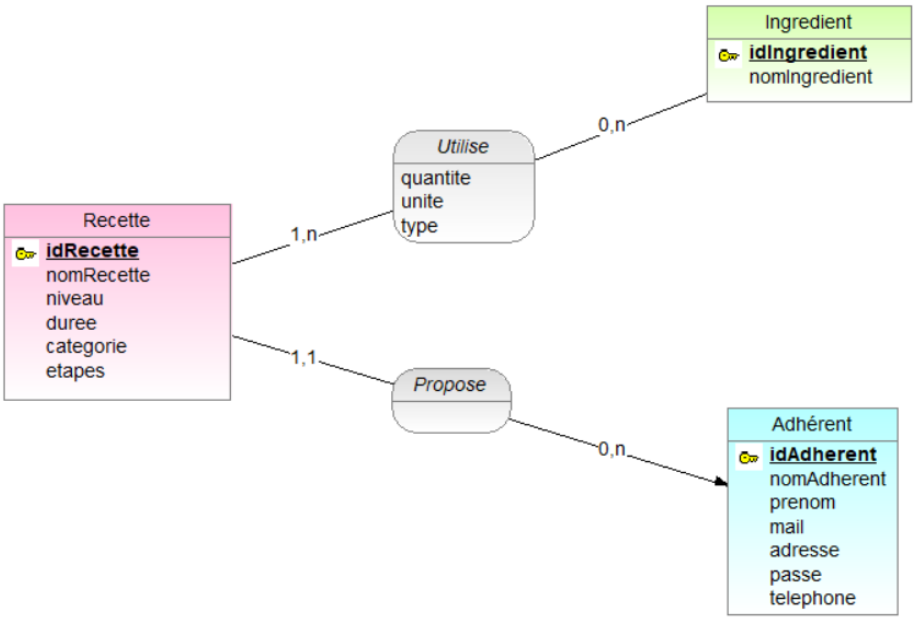


Schéma relationnel de la base de données du club de cuisine :

Adherent(*idAdherent, nomAdherent, prenom, mail, adresse, passe, telephone*) avec

- idAdherent : clé primaire

Recette(*idRecette, nomRecette, niveau, duree, categorie, etapes*) avec

- idRecette : clé primaire
- idAdherent : clé étrangère en référence à idAdherent de Adherent

Ingredient(*idIngredient, nomIngredient*) avec

- idIngredient : clé primaire

Utilise(*idRecette, idIngredient, quantite, unite, type*) avec

- idRecette, idIngredient : clé primaire
- idRecette : clé étrangère en référence à idRecette dans recette
- idIngredient : clé étrangère en référence à idIngredient dans ingredient

Dans ce contexte de base de données, on a :

- $(idIngredient, idRecette) \rightarrow nomIngredient$: à une valeur du couple $(idIngredient, idRecette)$ correspond une et une seule valeur de $nomIngredient$. Cependant, dans l'ensemble de ces deux attributs, à $idIngredient$ correspond également une seule valeur de $nomIngredient$.

→ La DF n'est donc **pas élémentaire**.

- $(idIngredient, idRecette) \rightarrow quantite$: à une valeur du couple $(idIngredient, idRecette)$ correspond une unique valeur de quantité. En revanche, à une valeur de $idIngredient$ ne correspond pas une et une seule valeur de $quantite$ car la valeur de $quantite$ dépend de la recette dans laquelle l'ingrédient identifié par $idIngredient$ intervient. De la même manière, à une valeur de $idRecette$ ne correspond pas une et une seule valeur de $quantite$ car la valeur de $quantite$ dépend de l'ingrédient qui intervient dans la recette identifiée par $idRecette$.

→ La DF est donc ici **élémentaire**.

b. Dépendance fonctionnelle directe

Définition

> **DF directe :**
>
> Soient A et B deux ensembles d'attributs tels que $A \rightarrow B$. La dépendance $A \rightarrow B$ est directe s'il n'existe pas un attribut C tel que $A \rightarrow C$ et $C \rightarrow B$.

En restant dans le contexte du club de cuisine :

- $idRecette \rightarrow nomAdherent$: à une valeur de l'attribut $idRecette$ ne correspond qu'une et une seule valeur de $nomAdherent$. En revanche, cette correspondance s'établit grâce aux DF $idRecette \rightarrow idAdherent$ puis $idAdherent \rightarrow nomAdherent$.

→ Ainsi, la DF $idRecette \rightarrow nomAdherent$ n'est **pas directe**.

- $idRecette \rightarrow idAdherent$: à une valeur de l'attribut $idRecette$ ne correspond qu'une et une seule valeur de $idAdherent$.

→ Ici, la DF est **directe**.

Astuce

> La dépendance fonctionnelle élémentaire est parfois abrégée en DFE. La dépendance fonctionnelle directe est parfois abrégée en DFD.

Maintenant que nous savons en quoi consiste une dépendance fonctionnelle et comment la caractériser (directe ou non, élémentaire ou non), entrons dans le vif du sujet : la normalisation.

2 La normalisation

La normalisation permet de contrôler les relations du schéma relationnel issu du MCD afin d'éviter les redondances sources d'anomalies de mise à jour, de permettre des recherches selon des critères variés et d'optimiser le fonctionnement informatique de la future base de données. Cette démarche peut donc conduire à une modification du schéma relationnel initial.

La normalisation impose cinq formes dites « normales », certaines pouvant présenter des variantes. Dans ce cours, nous nous focaliserons sur les trois premières formes normales qui sont suffisantes dans une grande majorité des cas.
Mais en quoi consistent ces formes normales ? Voyons plus en détail.

a. Première forme normale

Définition

> **Première forme normale :**
>
> Une relation est dite en première forme normale si, et seulement si, tout attribut qui la compose contient une valeur atomique. On la note 1FN.

À retenir

> Une valeur atomique (ou mono-valuée) est une valeur qui n'est pas décomposable en plusieurs parties, qu'elles soient ou non de même

nature.

- Observons la relation *Recette* de la base de données du club de cuisine. Son attribut « etapes » est décomposable (étape 1, étape 2, ...). Or, une étape est une action à laquelle est associée un rang qui permet de la positionner par rapport aux autres dans le but de les ordonner.

→ Notre relation *Recette* n'est donc pas conforme à la 1FN.

Si on ne la rend pas conforme, des difficultés vont se poser : comme toutes les recettes n'ont pas le même nombre d'étapes, on risque de sous-estimer ou surestimer la place à réserver dans la relation pour l'enregistrement des étapes dans un seul attribut.

- Étudions maintenant la relation *Adherent*. Son attribut « adresse » est décomposable (numéro, rue, complément, code postal et ville).

→ Notre relation *Adherent* n'est donc pas conforme à la 1FN.

Si on ne la rend pas conforme, on ne sera pas en mesure, par exemple, de recenser facilement tous les adhérents d'une même commune.

Il est peut-être parfois nécessaire de revoir le schéma relationnel de la base de données afin que ses relations se trouvent en 1FN.

Relation *Recette*

Pour modifier la relation *Recette*, il faut sortir l'attribut « etapes » de la relation *Recette* et créer une nouvelle relation comme suit :

Etape(*idRecette, rang, contenu*) avec
- idRecette, rang : clé primaire
- idRecette : clé étrangère en référence à idRecette de Recette

Astuce

Remarque :

On ne peut pas prendre uniquement le rang pour identifier une étape de recette de cuisine : le rang 1 est par exemple affecté à la première étape

de toutes les recettes. Il ne permet donc pas d'identifier à lui seul une et une seule étape d'une recette de cuisine donnée.

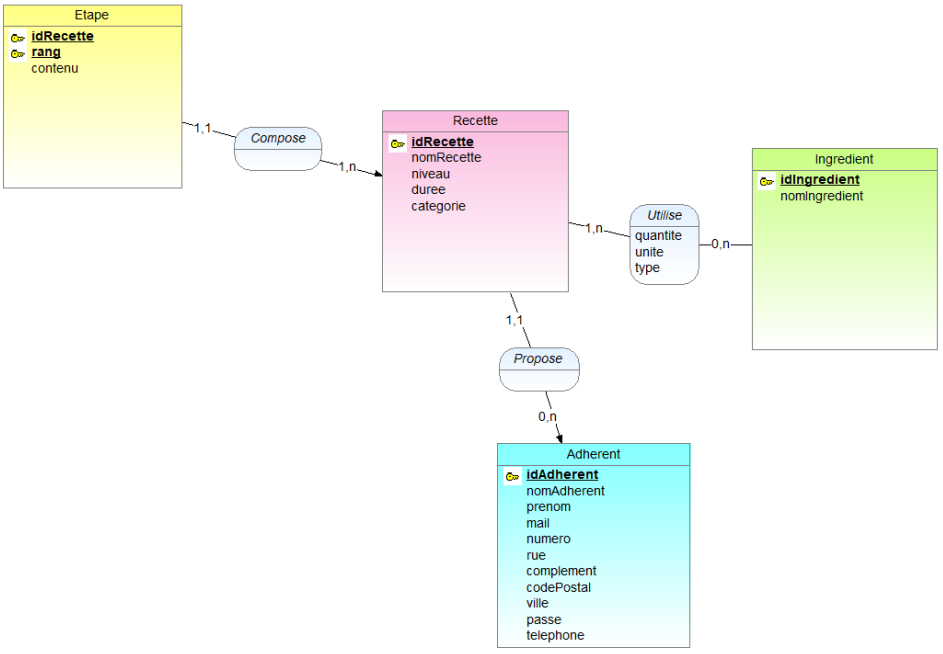
Relation *Adherent*

Pour modifier la relation *Adherent*, il faut décomposer l'attribut « adresse » en autant d'attributs que d'éléments qui le composent. La relation *Adherent* s'en voit donc ainsi modifiée :

Adherent(*idAdherent, nomAdherent, prenom, mail, numero, rue, complement, codePostal, ville, passe, telephone*) avec

idAdherent : clé primaire

On peut également répercuter ces modifications dans le MCD.



Une fois les relations du schéma mises en forme 1NF, il est temps de vérifier que le nouveau schéma relationnel respecte la deuxième forme normale.

b. Deuxième forme normale

Définition

Deuxième forme normale :

> Une relation est dite en deuxième forme normale si, et seulement si, elle est en première forme normale et si toutes les dépendances fonctionnelles entre la clé et les autres attributs sont élémentaires. On la note 2FN et on peut écrire : 1FN ⊂ 2FN.

Si nous regardons la relation

Utilise(*idRecette, idIngredient, quantite, unite, type*) avec
- idRecette, idIngredient : clé primaire
- idRecette : clé étrangère en référence à idRecette dans Recette
- idIngredient : clé étrangère en référence à idIngredient dans Ingredient

On a $idRecette, idIngredient \rightarrow type$, mais aussi $idIngredient \rightarrow type$. En effet, à un ingrédient identifié par « idIngredient » ne correspond qu'un et un seul type (viande, légume, condiment, boisson…).

→ La relation *Utilise* ne respecte donc pas la 2FN.

Si on ne la rend pas conforme, il va se produire inévitablement une redondance : l'information disant qu'un ingrédient donné relève d'un certain type **se répètera** dans la table.

Dans l'exemple ci-dessous, les informations stipulant respectivement que l'ingrédient 4 est un condiment et le 5 un légume se répètent.

idRecette	idIngredient	quantite	unite	type
1	2	200	grammes	viande
2	4	2	cuillère à café	condiment
3	5	500	grammes	légume
1	4	1	cuillère à soupe	condiment
5	5	1	kg	légume
4	10	50	mL	condiment

À retenir

> Une redondance d'information doit être évitée afin de limiter l'impact des mises à jour dans la base ainsi que les risques d'incohérence qui en découlent.

Pour y remédier, sachant que l'on a identifié $idIngredient \rightarrow type$, il semble naturel de sortir « type » de la relation *Utilise* pour l'ajouter aux attributs de la relation *Ingredient*.
On obtient :

Utilise(*idRecette, idIngredient, quantite, unite*) avec
- idRecette, idIngredient : clé primaire
- idRecette : clé étrangère en référence à idRecette dans Recette
- idIngredient : clé étrangère en référence à idIngredient dans Ingredient

Ingredient(*idIngredient, nomIngredient, type*) avec
- idIngredient : clé primaire

Maintenant que les relations du schéma respectent la 2NF, il ne nous reste plus qu'à contrôler que nos relations respectent la troisième forme normale.

c. Troisième forme normale

Définition

> **Troisième forme normale :**
>
> Une relation est dite en troisième forme normale si, et seulement si, elle est en seconde forme normale et si tout attribut n'appartenant pas à la clé primaire n'est pas en dépendance fonctionnelle directe avec un attribut ou un ensemble d'attributs non-clé. On la note 3FN et on peut écrire : 1FN $\subset$ 2FN $\subset$ 3FN.

En nous penchant sur la relation *Adherent*, on peut remarquer que $ville \rightarrow codePostal$. Or ni « ville », ni « codePostal » ne sont des attributs appartenant à la clé primaire de la relation *Adherent*.

→ Notre relation *Adherent* ne respecte donc pas la 3NF.

Si on ne la rend pas conforme, on se trouve encore une fois confronté à une redondance : les informations stipulant que le code postal de Guéret est 23000 et que le code postal de Brest est 29000 se répètent dans la table.

idAdherent	nomAdherent	prenom	mail	numero	rue
1	Dupont	Jacques	dup@orange.fr	3	rue blanc
2	Dampierre	Gérard	gege@free.fr	7	rue Dumc
3	Forestier	Denise	dd@gmail.com	9	rue au loups
4	Vielle	Rose	vr@sfr.fr	1	av. de prés
5	Javert	Jules	jj@sfr.fr	1	boule Brune
1	Duval	Isabelle	isa@bbox.fr	11bis	impas des hironc

Afin de corriger cela, sachant que l'on a mis en évidence $ville \rightarrow codePostal$, il suffit de sortir « codePostal » de la relation *Adherent* et de créer une nouvelle relation *Ville*.
Ce qui donne :

Adherent(*idAdherent, nomAdherent, prenom, mail, numero, rue, complement, idVille, passe, telephone*) avec

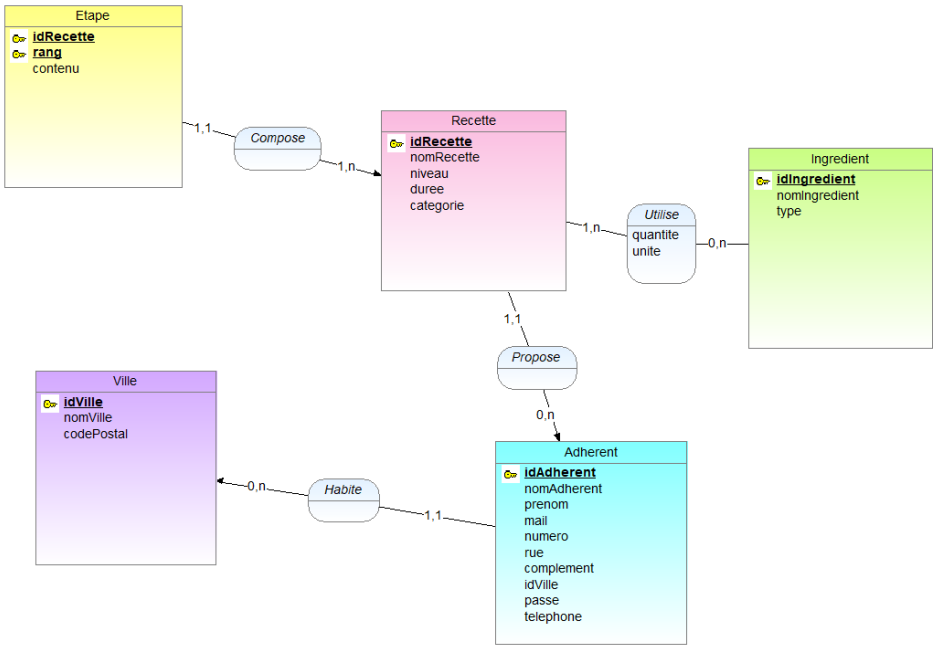
- idAdherent : clé primaire
- idVille : clé étrangère en référence à idVille dans Ville

Ville(*idVille, nomVille, codePostal*) avec

- idVille : clé primaire

On notera au passage la création d'un nouvel attribut « IdVille », que l'on définira comme numérique, préférant ainsi un attribut numérique à l'attribut alphabétique « nomVille » pour constituer une clé primaire.

Nous avons donc normalisé notre schéma relationnel en veillant à ce que ses relations respectent la 3NF, donc la 2NF et par conséquent la 1NF. Voici le MCD et le schéma relationnel final de notre exemple, maintenant normalisé.



Utilise(*idRecette, idIngredient, quantite, unite*) avec

- idRecette, idIngredient : clé primaire
- idRecette : clé étrangère en référence à idRecette dans Recette
- idIngredient : clé étrangère en référence à idIngredient dans Ingredient

Ingredient(*idIngredient, nomIngredient, type*) avec

- idIngredient : clé primaire

Adherent(*idAdherent, nomAdherent, prenom, mail, numero, rue, complement, idVille, passe, telephone*) avec

- idAdherent : clé primaire
- idVille : clé étrangère en référence à idVille dans Ville

Ville(*idVille, nomVille, codePostal*) avec

- idVille : clé primaire

Etape(*idRecette, rang, contenu*) avec

- idRecette, rang : clé primaire
- idRecette : clé étrangère en référence à idRecette de Recette

Recette(*idRecette, nomRecette, niveau, duree, categorie, etapes*) avec

- idRecette : clé primaire
- idAdherent : clé étrangère en référence à idAdherent de Adherent

À retenir

> Lorsque les relations du schéma relationnel respectent la 3FN, on peut considérer que la structure de la base de données est prête à être créée informatiquement.

Dans cette étape, on transmet le schéma relationnel au système informatique (nommé Système de Gestion de Base de Données, soit SGBD) qui aura pour charge de créer la structure puis de gérer la future base de données. Lors de cette étape, il est nécessaire de préciser des règles de fonctionnement au SGBD.

3 Les contraintes d'intégrité

Les **contraintes d'intégrité** sont des règles que doivent respecter les attributs des relations afin de **garantir la cohérence des données dans la base**. Elles doivent être précisées au moment où l'on implémente le schéma relationnel dans le système informatique chargé de créer la structure de la base. On en compte trois principales :

- la contrainte de domaine ;
- la contrainte de relation ;
- la contrainte de référence.

Voyons en quoi chacune de ces contraintes consistent.

a. La contrainte de domaine

Définition

Contrainte de domaine :

Lors de la création d'une table dans la base de données, il est nécessaire de préciser le domaine dans lequel chacun de ses attributs prend ses valeurs. C'est ce que l'on nomme une contrainte de domaine.

Il est important notamment de pouvoir comparer les valeurs d'un même attribut prises par plusieurs occurrences. Une comparaison n'est possible que si l'attribut est toujours enregistré avec le même format (on ne peut en effet pas comparer un A et un nombre !)..

Exemple

Dans la relation « etape » de la base de données du club de cuisine, si le rang de l'étape 1 est enregistré sous la forme du nombre « 1 » et si le rang de l'étape 2 est enregistré sous la forme alphanumérique « N° 2 », il sera difficile de les comparer, et donc de les ordonner.

→ Nous avons d'ailleurs commencé à réfléchir à cette contrainte de domaine dans le cours précédent, lorsque nous avons introduit notre dictionnaire de données dans lequel le type des attributs était défini.

Une fois la contrainte de domaine définie pour chaque attribut, le SGBD interdit l'introduction de tuples dans une relation dont le type d'un des attributs ne serait pas conforme à celui qui lui aura été attribué par la contrainte de domaine.

b. La contrainte de relation

Nous savons que toutes les relations du schéma relationnel doivent comporter une clé primaire, permettant d'identifier de manière unique les tuples qui y sont enregistrés.

Définition

> **Contrainte de relation :**
>
> La contrainte de relation impose que la clé primaire formée par un ou plusieurs attributs d'une relation soit unique et non nulle. C'est ce que l'on appelle la contrainte de relation.

C'est le respect de cette contrainte qui garantit l'identification sans ambiguïté d'un tuple dans une relation.

Ainsi, lors de la création d'une relation dans une base, **il est nécessaire de préciser cette contrainte de relation au SGBD**. Dès lors que cette contrainte de relation est précisée pour une table, le SGBD interdira l'introduction d'un tuple dont la valeur de la clé primaire sera identique à celle d'un tuple déjà présent dans la base ou sera nulle.

c. La contrainte de référence

Lorsque le schéma relationnel définit des relations faisant référence à des clés étrangères, cela soulève inévitablement une contrainte d'intégrité. Ainsi, si l'on insère un tuple dans une table faisant appel à des clés étrangères, le SGBD doit contrôler l'existence de ces clés dans les relations d'où elles proviennent.

Exemple

> Dans la base de données du club de cuisine, considérons l'insertion d'un tuple dont les attributs « idIngredient » et « idRecette » sont respectivement 12 et 27. Le SGBD n'acceptera cette insertion qu'à condition que l'ingrédient dont l'attribut $idIngredient = 12$ existe dans la table « ingredient » et que la recette dont l'$idRecette = 27$ existe dans la table « recette ».

De plus, il est nécessaire d'indiquer au SGBD ce qu'il doit faire dans la cadre de la suppression d'un tuple dans une table dont la clé primaire est étrangère dans une autre table. Plusieurs options sont possibles :

- interdire la suppression ;
- supprimer également les tuples concernés dans les autres tables ;
- avertir l'utilisateur d'une incohérence ;
- mettre les valeurs des attributs concernés à une valeur nulle dans les tuples concernés, si l'opération est possible (ce n'est pas le cas si la clé primaire du tuple supprimé au départ intervient en clé étrangère dans la clé primaire d'une autre table).

 Définition

Intégrité référentielle :

L’intégrité référentielle est une situation dans laquelle pour chaque tuple d'une table A qui fait référence à un tuple d'une table B, le tuple ainsi référencé existe réellement dans la table B.

Conclusion :

Nous sommes maintenant en mesure de mettre en œuvre une base de données en transmettant au SGBD un schéma relationnel préalablement élaboré ainsi que les contraintes d'intégrité qui l'accompagnent. Dans le cours qui suit, nous nous intéresserons de plus près aux missions des SGBD chargés de mettre en œuvre informatiquement les bases de données.