Analyse et Conception d'une base de données

I. Définition d'une Base de Données

En Informatique une donnée est un élément essentiel voir même indispensable pour la réalisation d'un traitement quelconque. Elle est la représentation d'une information sous forme conventionnelle destinée à faciliter son traitement.
Donc une base de données est une collection d'informations structurées de manière ordonnées et stockées sur un support physique à savoir une clé USB, un disque dur…

Ce terme est apparu pour la première fois en 1964 et elle se conçoit toujours en fonction d'un thème ou d'une activité.
Prenons l'exemple d'une société de vente et de location de voitures, sa base de données devrait contenir l'ensemble des clients et des employés qui travaille au sein de cet entreprise (noms, prénoms, âges, matricules, salaires…), l'ensemble des voitures à leur disposition (plaques d'immatriculations, modèles, prix, année de fabrication…) etc.

II. Qu'est-ce qu'un SGBD ?

Un **SGBD (S**ystème de **G**estion de **B**ase de **D**onnées**)** est un type de logiciel permettant de gérer une base de données.
Parmi les SGBD on peut citer :

- Microsoft Access
- Oracle
- SQL Server
- MySQL
- PostgreSQL

Le SGBD a pour but d'offrir à l'utilisateur une intéraction avec la base de données sous forme d'un dialogue pour rechercher, sélectionner et modifier les données.
On distingue deux catégories d'utilisateurs :

- Administrateur : celui qui a le control total et gère le fonctionnement de la base.
- L'hôte ou le client : qui peut utiliser la base cependant avec certaines restrictions.

Par conséquent il existe deux façons pour interroger la base :

- Soit avec les programmes d'application (Langage de Programmation)
- Soit avec les langages de requêtes (Langage SQL)

1.

III. Les différents modèles de conception

Analyse et Conception d'une base de données

a) Qu'est-ce qu'un modèle ?

Un modèle est une représentation abstraite de la réalité, c.-à-d. elle est proche de cette dernière. Elle est conçue pour avoir un point de départ, une référence initiale

b) Le Modèle Conceptuel de Données (MCD)

Le modèle conceptuel des données est une représentation statique du système d'information de l'entreprise qui met en évidence sa sémantique. Il a pour but d'écrire de façon formelle les données qui seront utilisées par le système d'information. Il s'agit donc d'une représentation des données, facilement compréhensible. Le formalisme adopté par la méthode Merise pour réaliser cette description est basé sur les concepts « entité / association ».

Un système d'information (SI) est un ensemble organisé de ressources (matériels, logiciels, personnel, données et procédures) qui permet de collecter, regrouper, classifier, traiter et diffuser de l'information sur un environnement donné

1) Qu'est-ce qu'une entité ?

Une entité est un objet, une chose concrète ou abstraite qui peut être reconnue distinctement.

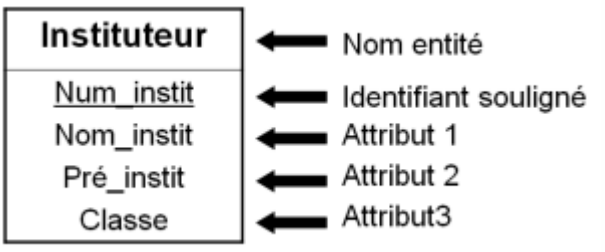
Un ensemble d'entité est la représentation commune que l'on adopte pour des entités qui possèdent les mêmes caractéristiques.

Une entité est une occurrence d'un ensemble d'entités. Ex :

Une personne (CLIENT)

Un lieu (DEPOT, BUREAU, ATELIER, …)

Un objet documentaire(LIVRE, OUVRAGE, DOSSIER,…)



2) Qu'est-ce qu'un identifiant ?

Analyse et Conception d'une base de données

C'est une propriété particulière de l'entité, une représentation de l'une des occurrences de l'entité ou de l'association. Le meilleur moyen pour ne pas risquer d'avoir des synonymes consiste à prendre des numéros de références comme identifiant. Un identifiant peut être simple c.à.d. constitué d'une seule propriété élémentaire (NUM_ELEV)

3) Qu'est-ce qu'une occurrence ?

Une occurrence est une représentation d'une entité.

4) Qu'est-ce qu'un attribut ?

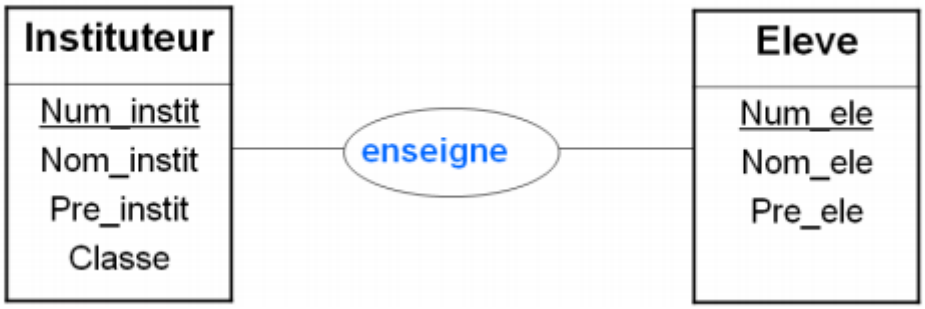
Un attribut est une propriété d'une entité (une caractéristique associée à une entité).

◦ Exemple: l'âge ou le prénom d'une personne, la couleur d'une voiture, la

5) Qu'est-ce qu'une association ?

Une association est un lien entre plusieurs entités.

Représentation graphique d'une association:



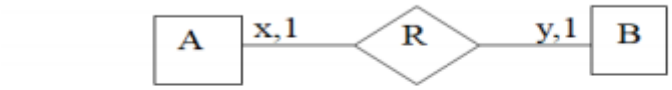
6) Qu'est-ce qu'une cardinalité ?

Une cardinalité est le nombre de fois qu'une entité participe à une association avec une autre (en réalité c'est le nombre de fois minimal et maximal qu'une occurrence d'une des entités associée peut intervenir dans l'association).

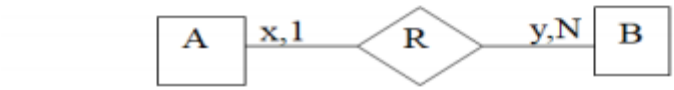
Elle est notée (min , max).

Analyse et Conception d'une base de données

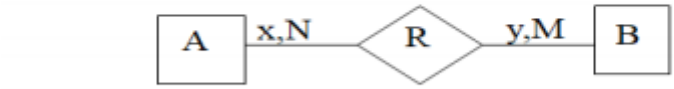
Cardinalités d'associations Typologie des associations binaires



Association de type 1-1 (*one-to-one*)



Association de type 1-N (*one-to-many*)

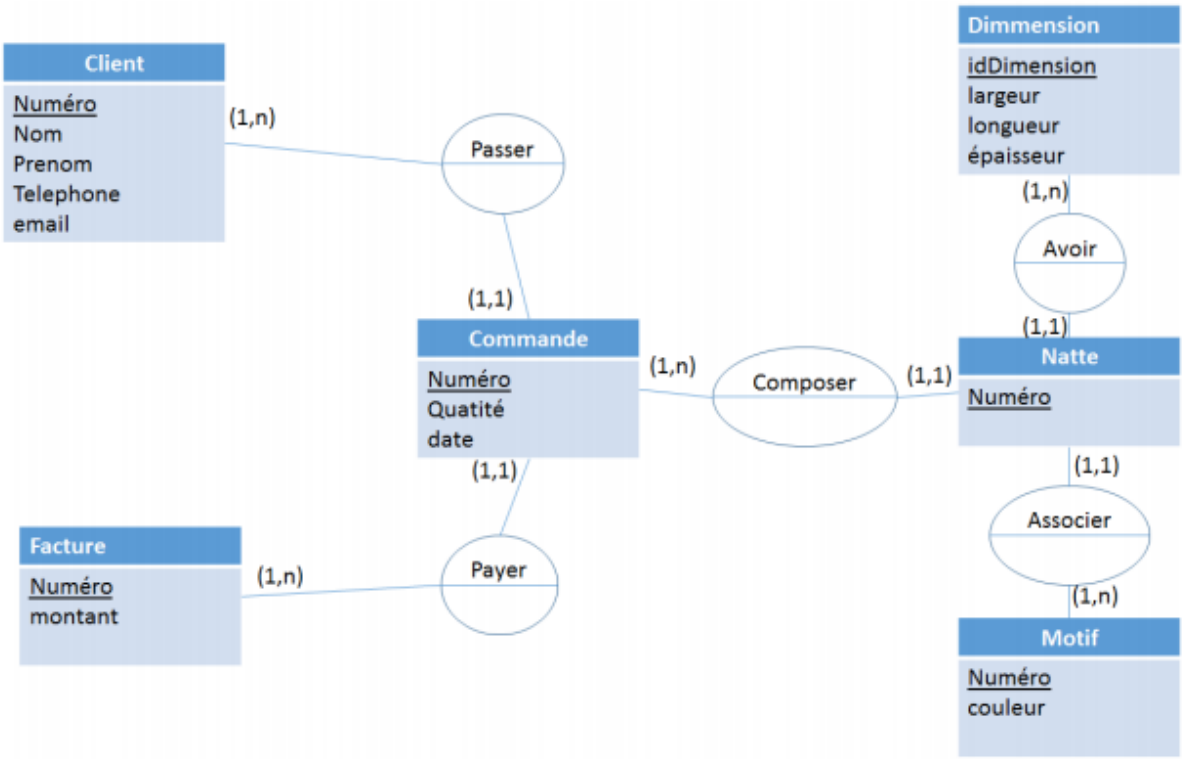


Association de type N-M (*many-to-many*)

Ex : un client peut commander entre 1 et n produits (1 , n).

7) Exemple de MCD

Analyse et Conception d’une base de données



c) Le Modèle Logique de Données (MLD)

Le **Modèle Logique de Données** (MLD) **est** la modélisation **logique** des **données** qui tient compte du niveau organisationnel des **données**. Il s'agit d'une vue **logique** en terme d'organisation de **données** nécessaire à un traitement. En symbolique, il y a des flèches entre les tables au lieu des patates entre les entités du MCD.

La transcription d'un MCD en modèle relationnel s'effectue selon quelques règles simples qui consistent d'abord à transformer toute entité en table, avec l'identifiant comme clé primaire, puis à observer les valeurs prises par les cardinalités maximum de chaque association pour représenter celle-ci soit (ex : card. max 1 [1-**1** ou 0-**1**]) par l'ajout d'une **clé étrangère** dans une table existante, soit (ex : card. max n [0-**N** ou 1-**N**]) par la création d'une nouvelle table dont la clé primaire est obtenue par concaténation de clés étrangères correspondant aux entités liées

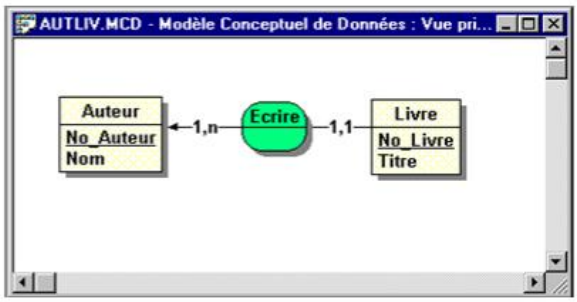
Un MLD est essentiellement composé de tables logiques reliées entre elles par des flèches.

Analyse et Conception d'une base de données

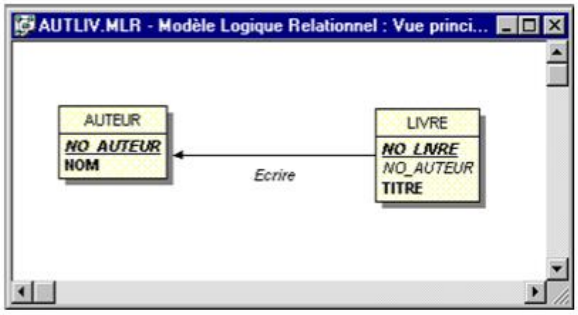
Un MLD est essentiellement composé de tables logiques reliées entre elles par des flèches.

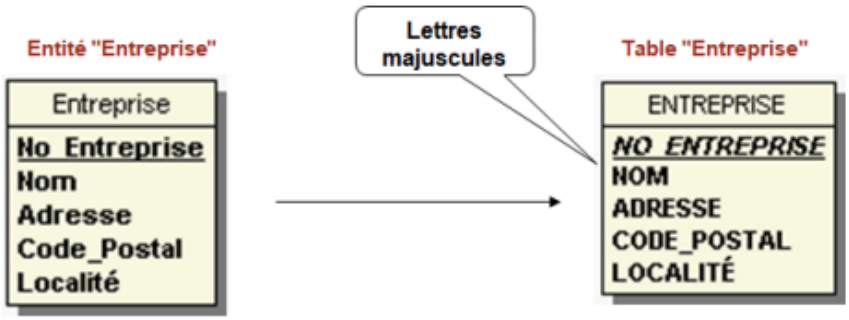
Exemple

MCD



MLD





1) **Règles de passage du MCD au MLD**

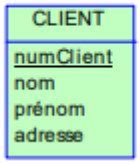
- **Règle numéro 1** :

Une entité du MCD devient une relation, c'est à dire une table. Dans un SGBD (Système de Gestion de base de données) de type relationnel, une table est une structure tabulaire dont chaque ligne correspond aux données d'un objet enregistré (d'où le terme enregistrement) et où chaque colonne correspond à une propriété de cet objet. Une table contiendra donc un ensemble d'enregistrements. Une ligne correspond à un enregistrement. Une colonne correspond à un champ. La valeur prise par un champ pour un enregistrement donné est située à l'intersection ligne-colonne correspondant à enregistrement-champ. Il n'y a pas de limite théorique au nombre d'enregistrements que peut contenir une table. Par contre, la limite est liée à l'espace de stockage.

Son identifiant devient la clé primaire de la relation. La clé primaire permet d'identifier de façon unique un enregistrement dans la table. Les valeurs de la clé primaire sont donc uniques. Les valeurs

de la clé primaire sont obligatoirement non nulles. Dans la plupart des SGBDR (Système de Gestion de Base de Données Relationnelle), le fait de définir une clé primaire donne lieu automatiquement à la création d'un index. Un index est un fichier interne au SGBD. L'utilisateur standard n'a pas besoin d'y accéder. L'index a pour but d'accélérer les traitements de recherche, de tri, de filtre et notamment sur les tables avec de nombreux enregistrements. La contrepartie est que l'index nécessite de l'espace mémoire et surtout, les temps d'insertion, de suppression d'enregistrements sont plus importants car il faut mettre à jour à la fois la table et l'index.

Exemple :



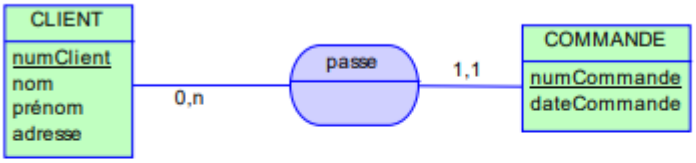
CLIENT(numClient , nom , prenom , adresse)
numClient : clé primaire de la table CLIENT

numClient	Nom	Prenom	adresse
1	Dupont	Pierre	5 rue de Paris 93000 Saint-Denis
2	Durand	Raymond	68 rue Alphonse Daudet 77540 Noisy le grand
3	Dupuis	Elisa	1, boulevard Louis Blériot 94800 Villejuif
4	Dubois	Raymonde	15bis, rue de la Gaité 75014 Paris
...	...	...	...

- **Règle numéro 2** :
  Une association de type 1:N (c'est à dire qui a les cardinalités maximales positionnées à « 1 » d'une côté de l'association et à « n » de l'autre côté) se traduit par la création d'une clé étrangère dans la relation correspondante à l'entité côté « 1 ». Cette clé étrangère référence la clé primaire de la relation correspondant à l'autre entité.

Analyse et Conception d'une base de données

Exemple :



CLIENT(numClient , nom , prenom , adresse)
numClient : clé primaire de la table CLIENT
COMMANDE(numCommande ,dateCommande , #numClient)
numCommande : clé primaire de la table COMMANDE
#numClient : clé étrangère qui référence numClient de la table CLIENT

Table CLIENT :

numClient	Nom	Prenom	adresse
1	Dupont	Pierre	5 rue de Paris 93000 Saint-Denis
2	Durand	Raymond	68 rue Alphonse Daudet 77540 Noisy le grand
3	Dupuis	Elisa	1, boulevard Louis Blériot 94800 Villejuif
4	Dubois	Raymonde	15bis, rue de la Gaité 75014 Paris
...	...	...	...

Table COMMANDE :

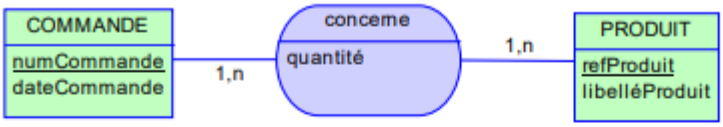
numCommande	dateCommande	numClient
11	1/02/2014	1
62	1/02/2014	3
423	2/02/2014	3
554	3/02/2014	2
...	...	...

Même si les tables COMMANDE et CLIENT sont 2 tables distinctes, il est possible de retrouver toutes les informations des 2 tables de façon cohérente grâce à la clé étrangère. Exemple de questions auxquelles il est possible de répondre : - Quel est le nom du client qui a passé la commande 11 ? - Quels sont les noms des clients qui ont commandé le 1/02/2014 ? - Combien de commandes a passé Elisa Dupuis ? - Quelle est l'adresse du client qui a passé la commande 423 ?

- **Règle numéro 3** :
  Une association de type N :N (c'est à dire qui a les cardinalités maximales positionnées à « N » des 2 côtés de l'association) se traduit par la création d'une table dont la clé primaire est composée des clés étrangères référençant les relations correspondant aux entités liées par l'association. Les éventuelles propriétés de l'association deviennent des attributs de la relation.

Analyse et Conception d'une base de données

Exemple :



COMMANDE(numCommande ,dateCommande)
numCommande : clé primaire de la table COMMANDE
PRODUIT(refProduit, libelleProduit)
refProduit : clé primaire de la table PRODUIT
CONCERNE(#numCommande , #refProduit , quantité)
#numCommande , #refProduit : clé primaire composée de la table CONCERNE
#numCommande : clé étrangère qui référence numCommande de la table COMMANDE
#refProduit : clé étrangère qui référence refProduit de la table PRODUIT

Si le nom du MCD n'est pas significatif, on peut renommer le nom de la table.
Dans notre exemple, plutôt que d'appeler la table « CONCERNE », on la nommera « LIGNE_DE_COMMANDE ».

LIGNE_DE_COMMANDE (#numCommande , #refProduit , quantité)
#numCommande , #refProduit : clé primaire composée de la table CONCERNE
#numCommande : clé étrangère qui référence numCommande de la table COMMANDE
#refProduit : clé étrangère qui référence refProduit de la table PRODUIT

Table COMMANDE :

numCommande	dateCommande
11	1/02/2014
62	1/02/2014
423	2/02/2014
554	3/02/2014
...	...

Table PRODUIT :

refProduit	libelleProduit
C24	Chocolat
B12	Bière
L22	Lait
...	...

TableLIGNE_DE_COMMANDE :

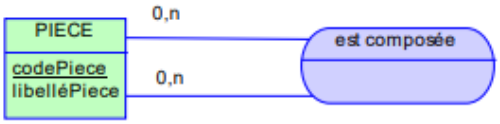
numCommande	refProduit	Quantite
11	C24	3
62	B12	3
62	C24	8
423	C24	8765
...	...	...

Analyse et Conception d'une base de données

Associations ternaires : Les règles définies ci-dessus s'appliquent aux associations ternaires.

Associations réflexives : Les règles définies ci-dessus s'appliquent aux associations réflexives.

Exemple :



PIECE(codePiece ,libellePiece)
COMPOSITION(refProduit, libelleProduit)
COMPOSITION (#pieceComposee , #pieceComposante)

IV. LE LANGAGE SQL

SQL (sigle de *Structured Query Language*, en français **langage de requête structurée**) est un langage informatique normalisé servant à exploiter des bases de données relationnelles. La partie *langage de manipulation des données* de SQL permet de rechercher, d'ajouter, de modifier ou de supprimer des données dans les bases de données relationnelles.

Outre le langage de manipulation des données, la partie *langage de définition des données* permet de créer et de modifier l'organisation des données dans la base de données, la partie *langage de contrôle de transaction* permet de commencer et de terminer des transactions, et la partie *langage de contrôle des données* permet d'autoriser ou d'interdire l'accès à certaines données à certaines personnes.

Créé en 1974, normalisé depuis 1986, le langage est reconnu par la grande majorité des systèmes de gestion de bases de données relationnelles (abrégé SGBDR) du marché.

langage de manipulation de données

Les instructions de manipulation du contenu de la base de données commencent par les mots clés *SELECT*, *UPDATE*, *INSERT* ou *DELETE* qui correspondent respectivement aux opérations de recherche de contenu, modification, ajout et suppression. Divers mots-clés tels que *FROM*, *JOIN* et *GROUP BY* permettent d'indiquer les opérations d'algèbre relationnelle à effectuer en vue d'obtenir le contenu à manipuler.

langage de définition de données

Les instructions de manipulation des métadonnées - description de la structure, l'organisation et les caractéristiques de la base de données - commencent avec les mots-clés *CREATE*, *ALTER*, *DROP*, *RENAME*, *COMMENT* ou *TRUNCATE* qui correspondent aux opérations d'ajouter, modifier, supprimer, renommer, commenter ou vider une métadonnée. Ces mots clés sont immédiatement suivis du type de métadonnée à manipuler - *TABLE*, *VIEW*, *INDEX*...

langage de contrôle de données et langage de contrôle des transactions

Les mots clés *GRANT* et *REVOKE* permettent d'autoriser des opérations à certaines personnes, d'ajouter ou de supprimer des autorisations. Tandis que les mots clés *COMMIT* et *ROLLBACK* permettent de confirmer ou annuler l'exécution de transactions.

La syntaxe de SQL fait l'objet de la norme *ISO 9075*. Cette norme laisse la possibilité aux producteurs de SGBD d'y ajouter des instructions spécifiques et non normalisées. La norme a évolué au cours des années en vue de s'adapter aux demandes, et les éditeurs de SGBD ont souvent ajouté des possibilités à leurs produits avant que celles-ci fassent objet de normes[5], ce qui provoque des variations dans la

Analyse et Conception d'une base de données

compréhension et l'interprétation qui est faite d'un code source en SQL par les différents logiciels de SGBD6. Ces différences font qu'un code source écrit sans précautions pour un SGBD donné ne fonctionnera pas forcément avec un autre SGBD.

Manipulation de données

Le Langage de manipulation de données LMD, soit Data Manipulation Language, DML, en anglais, est un sous-ensemble du SQL utilisé pour ajouter, modifier, et supprimer des données :

- `INSERT` insère des n-uplets (informellement appelés lignes et appelés *tuples* en anglais) dans une table existante, exemple :

```
INSERT INTO a_table (field1, field2, field3)
  VALUES ('test', 'N', NULL);
```

- `UPDATE` Modifie un ensemble de n-uplets existant dans une table, exemple :

```
UPDATE a_table
  SET field1 = 'updated value'
WHERE field2 = 'N';
```

- `DELETE` Supprime un ensemble de n-uplets existant dans une table, exemple :

```
DELETE FROM a_table
  WHERE field2 = 'N';
```

- `MERGE` Combine les données de plusieurs tables. C'est la combinaison de `INSERT` et `UPDATE`. Il peut être nommé `UPSERT`, `INSERT OR REPLACE INTO`, ou encore `INSERT ON DUPLICATE KEY UPDATE` dans certains moteurs de base de données.

```
MERGE INTO table_name USING table_reference ON (condition)
 WHEN MATCHED THEN
 UPDATE SET column1 = value1 [, column2 = value2 ...]
 WHEN NOT MATCHED THEN
 INSERT (column1 [, column2 ...]) VALUES (value1 [, value2 ...
```

Analyse et Conception d’une base de données

SELECT...FROM...WHERE

Si on devait se fier uniquement à son nom, l'instruction SELECT devrait effectuer une sélection. Dans la réalité du SQL, elle est plus puissante et permet de faire aussi une projection, un ou plusieurs produit cartésien, et quelques autres opérations annexes. La syntaxe de l'opération SELECT est relativement complexe : elle est composée de plusieurs clauses, qui s'écrivent ainsi :

```
SELECT colonne
FROM table
WHERE condition
ORDER BY attribut
GROUP BY attribut
HAVING condition
LIMIT nombre_de_lignes ;
```

SELECT

Dans le cas le plus simple, la syntaxe de SELECT est la suivante :

```
SELECT nom_colonnes : /* les noms des colonnes sont séparés par des virgules */
FROM nom_table ;
```

Le SELECT effectue une projection : les noms des colonnes indiqués après le SELECT correspondent aux colonnes qui seront conservées, les autres étant éliminées par la projection. Le FROM indique dans quelle

table il faut effectuer la projection et la sélection. On peut se limiter à faire une projection, ce qui fait que la syntaxe suivante est parfaitement possible :

```
SELECT nom, prénom
FROM personne ;
```

Doublons

On peut préciser qu'il est possible que certaines lignes donnent des doublons dans la table. Par exemple, rien n’empêche que plusieurs personnes différentes aient le même nom : une projection de la colonne nom dans une table de personne donnera fatalement quelques doublons. Mais il est possible d'éliminer ces doublons en utilisant un mot-clé. Ce mot-clé DISTINCT doit être placé entre le SELECT et les noms des colonnes.

```
SELECT DISTINCT nom_colonnes
FROM nom_table ;
```

Par exemple, prenons cette table :

```
CREATE TABLE Animal (
   id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
   espece VARCHAR(40) NOT NULL,
   sexe CHAR(1),
   date_naissance DATETIME NOT NULL,
   nom VARCHAR(30),

   PRIMARY KEY (id)
)
```

Voici la requête SQL pour obtenir une table dans laquelle on ne récupérerait que les noms des animaux (sans doublons) :

```
SELECT DISTINCT nom
FROM Animal
```

Voici la requête SQL pour obtenir une table dans laquelle on ne récupérerait que les noms et les dates de naissance des animaux (sans doublons) :

```
SELECT DISTINCT nom , date_naissance
FROM Animal
```

Omettre la projection

Si jamais vous souhaitez conserver toutes les colonnes lors d'une projection, vous devez utiliser la syntaxe suivante :

```
SELECT *
FROM Nom_table
WHERE condition
```

Par exemple, la requête SELECT suivante sélectionne toutes les lignes de la table personne où l'attribut prenom vaut "Jean", sans faire de projection.

```
SELECT *
FROM personne
WHERE (prenom = "Jean")
```

Agrégations

Dans le SELECT, il est possible d'utiliser certaines fonctions qui permettent respectivement de :

- faire la somme de tous les attributs d'une colonne (leur somme) ;
- déterminer quelle est la plus petite valeur présente dans la colonne (son minimum) ;
- déterminer quelle est la plus grande valeur présente dans la colonne (son maximum) ;
- déterminer quelle est la moyenne de la colonne ;
- déterminer quelle est le nombre de lignes de la colonne projetée.

Ces fonctions sont respectivement nommées :

- SUM (somme d'une colonne) ;
- AVG (moyenne d'une colonne) ;
- MIN (minimum d'une colonne) ;
- MAX (maximum d'une colonne) ;
- COUNT (nombre de lignes d'une colonne).

Par exemple, imaginons que vous ayez une table Personne contenant une liste de personnes, avec un attribut/colonne age et un autre attribut pour le nom. Supposons que vous souhaitiez calculer la moyenne d'âge des personnes de cette liste : vous pouvez le faire en utilisant cette requête :

```
SELECT AVG(age)
FROM Personne ;
```

Comme autre exemple, vous pouvez prendre la personne la plus âgée en utilisant la requête suivante :

```
SELECT MAX(age)
FROM Personne ;
```

De même, vous pouvez savoir combien de noms différents contient la table en utilisant cette requête :

```
SELECT DISTINCT COUNT(nom)
FROM Personne ;
```

WHERE

Si on veut effectuer une sélection, c'est cette syntaxe qu'il faut utiliser

```
SELECT nom_colonnes
FROM nom_table
WHERE condition_selection ;
```

Analyse et Conception d'une base de données

Le WHERE, facultatif, permet de faire une sélection : il précise la condition que les lignes doivent respecter pour être conservées dans la table résultat.

Comparaisons de base

Les tests autorisés en SQL sont les conditions les plus basiques :

- a < b ;
- a > b ;
- a <= b ;
- a >= b ;
- a = b ;
- a <> b.

De plus, les opérateurs booléens AND, OR et NOT sont possibles (avec le XOR en plus).

Les comparaisons vues plus haut ne permettent pas de vérifier si le contenu d'une colonne est NULL. Ainsi, une condition du type age = NULL ne sera tout simplement pas accepté par le SGBD. Pour vérifier si une colonne contient ou non la valeur NULL, il faut utiliser des conditions spéciales, notées respectivement :

- IS NULL ;
- IS NOT NULL.

Pour donner un exemple, la requête suivante renvoie toutes les personnes dont l'âge est inconnu :

```
SELECT *
FROM Personne
WHERE age IS NULL ;
```

Comme autre exemple, la requête suivante renvoie toutes les personnes dont l'adresse est connue :

```
SELECT *
FROM Personne
WHERE adresse IS NOT NULL ;
```

Condition intervallaire

Il est aussi possible de vérifier si tel attribut est compris dans un intervalle bien précis. Pour cela, la condition s'écrit comme ceci :

```
SELECT ...
FROM ...
WHERE attribut BETWEEN minimum AND maximum ;
```

Par exemple, cette requête renvoie les lignes de la table personne qui comprend les personnes dont l'âge est compris entre 18 et 62 ans (les personnes en âge de travailler) :

```
SELECT *
FROM Personne
WHERE age BETWEEN 18 AND 62 ;
```

Cela fonctionne aussi avec les chaines de caractère ou les dates. Par exemple, la requête suivante renvoie toutes les personnes qui sont nées entre le 2 juillet 2000 et le 3 octobre 20014 :

```
SELECT *
FROM Personne
WHERE date_naissance BETWEEN '2000-07-02' AND '2014-10-3' ;
```

La condition inverse, qui vérifie que l'attribut n'est pas dans l'intervalle existe aussi : c'est la condition NOT BETWEEN. Elle s'utilise comme la condition BETWEEN. Par exemple, cette requête renvoie les lignes de la table personne qui comprend les personnes dont l'âge n'est pas compris entre 18 et 62 ans (les personnes en âge de travailler) :

```
SELECT *
FROM Personne
WHERE age NOT BETWEEN 18 AND 62 ;
```

Exemples

Prenons maintenant la table PERSONNE définie au-dessus.

Voici comment récupérer les personnes dont on connait l'âge :

```
SELECT *
FROM personne
WHERE (age IS NOT NULL) ;
```

Voici comment récupérer les personnes dont on ne connait pas la taille :

```
SELECT *
FROM personne
WHERE (taille IS NULL) ;
```

Maintenant, prenons la table suivante :

```
create table PERSONNE
(
   ID_PERSONNE int not null ,
   NOM varchar not null ,
   PRENOM varchar not null ,
   AGE int ,
   CATEGORIE_PROFESSIONNELLE varchar ,
   NOMBRE_ENFANTS int ,
   TAILLE float ,

   primary key (ID_PERSONNE) ,
) ;
```

Voici la requête pour récupérer toutes les personnes dont la taille est supérieure à 170 centimètres :

```
SELECT *
```

```
FROM personne
WHERE (taille > 170) ;
```

Et voici la requête pour récupérer les noms et prénoms des personnes qui ont la majorité (dont l'âge est égal à 18 ans) :

```
SELECT nom, prénom
FROM personne
WHERE (age = "18") ;
```

ORDER BY

On peut demander au SGBD de trier les données dans la table résultat, que ce soit dans l'ordre croissant ou dans l'ordre décroissant. Pour cela, il faut utiliser l'instruction ORDER BY, juste en dessous du WHERE. Cette instruction ORDER BY a besoin de plusieurs informations pour fonctionner :

- quelles sont les colonnes à prendre en compte lors du tri ;
- dans quel ordre les prendre en compte ;
- faut-il trier chaque colonne par ordre croissant ou décroissant.

Pour cela, le ORDER BY est suivi d'un ou de plusieurs noms de colonne. De plus, chaque nom de colonne est suivi d'un mot-clé qui indique s'il faut trier dans l'ordre croissant ou décroissant : ces mot-clés sont respectivement les mot-clés ASC et DSC (pour Ascendant et Descendant).

Par exemple, la requête suivante sélectionne les personnes majeures de la table Personne, les personnes de la table résultat étant triée de l'âge le plus petit vers l'âge le plus grand.

```
SELECT *
FROM Personne
WHERE age > 18 AND age IS NOT NULL
ORDER BY age ASC ;
```

Par contre, la requête suivante trie les résultats par âge décroissant.

```
SELECT *
FROM Personne
WHERE age > 18 AND age IS NOT NULL
ORDER BY age DSC ;
```

FROM

Maintenant que nous avons étudié en détail le SELECT et le WHERE, nous allons nous pencher plus en détail sur le FROM. Dans les exemples précédents, nous avons vu que le FROM permet de préciser la table sur laquelle nous voulons effectuer la requête. Mais dans les fait, le FROM est aussi plus puissant que prévu : il permet aussi d'effectuer des produits cartésiens entre plusieurs tables et des jointures. Il permet aussi d'effectuer ce qu'on appelle des sous-requêtes. Tout ce qu'il faut retenir, c'est que l'expression qui suit le FROM doit avoir pour résultat une table, cette table étant celle sur laquelle on effectue la requête.

Produit cartésien

Analyse et Conception d'une base de données

Effectuer un produit cartésien est relativement simple : il suffit d'indiquer plusieurs tables, séparées par des virgules à la suite du FROM. Par exemple, cette requête effectue un produit cartésien des tables Personne et Emploi :

```
SELECT *
FROM Personne , Emploi ;
```

Jointures

On peut parfaitement créer les jointures à la main, en décrivant le produit cartésien dans le FROM et en mettant la condition de la jointure dans le WHERE. Mais le langage SQL fournit une syntaxe spéciale pour les jointures. Mieux : il fournit plusieurs types de jointures, qui différent sur quelques points de détail. La syntaxe la plus simple est la jointure normale, aussi appelé jointure interne, ou encore INNER JOIN.

```
SELECT *
FROM table_1 INNER JOIN table_2 ON condition
WHERE ... ;
```

La jointure naturelle est un cas particulier de jointure interne, qui correspond au cas où la condition vérifie l'égalité de deux colonnes et où les deux colonnes en question ont le même nom de colonne. Avec ces jointures, il n'y a pas besoin de préciser la condition que doivent respecter les deux tables, celle-ci étant implicite : c'est l'égalité des deux colonnes qui ont le même nom dans les deux tables. Ces jointures naturelles s'écrivent avec le mot-clé NATURAL JOIN, qui sépare les deux tables :

```
SELECT *
FROM table_1 NATURAL JOIN table_2
WHERE ... ;
```

A coté de cette jointure interne, il existe des jointures externes, qui ajoutent des lignes pour lesquelles la condition n'est pas respectée. Dans tous les cas, les lignes ajoutées voient leurs colonnes vides remplies avec la valeur NULL (la table résultat a plus de colonnes que les tables initiales). Il existe trois grands types de jointures externes :

- la plus commune est la jointure gauche, dans laquelle les lignes de la première table sont ajoutées dans la table résultat, même si elles ne respectent pas la condition ;
- la jointure droite est similaire, sauf que les lignes de la seconde table remplacent les lignes de la première table ;
- la jointure totale peut être vue comme un mélange d'une jointure gauche et droite : les lignes des deux tables sont copiées, même si elles ne respectent pas la condition et les colonnes en trop sont remplies avec NULL.

Les mot-clés pour ces jointures sont respectivement :

- LEFT JOIN ;
- RIGHT JOIN ;
- FULL JOIN.

GROUP BY

Analyse et Conception d'une base de données

La clause GROUP BY permet de grouper plusieurs lignes en une seule, à la condition que ces lignes aient un attribut/colonne identique. Si on se contente de faire une projection sur la colonne identique dans les lignes, la clause GROUP BY élimine les doublons. On peut se demander à quoi cela peut servir, vu que le mot-clé distinct permet de faire exactement la même chose. Mais la différence apparait quand on utilise des fonctions comme MAX, MIN, AVG, SUM ou COUNT : ces fonctions n'agissent plus sur une colonne complète, mais sur un groupe à la fois.

Prenons un exemple classique, avec une table ACHAT qui mémorise des informations sur des achats :

- quel client a effectué l'achat : clé étrangère client ;
- quel est le montant de l'achat : attribut montant, de type INT ;
- et d'autres dont on se moque pour cet exemple.

Il se trouve qu'un client peut faire plusieurs achats, à des jours différents, ou acheter des articles différents en une fois. Les clauses GROUP BY et une projection bien choisie nous permettent de calculer quel montant a dépensé le client dans le magasin. Pour cela, il suffit de regrouper toutes les lignes qui font référence à un même client avec un GROUP BY client. Une fois cela fait, on ajoute une fonction SUM dans la projection, afin de sommer les montants pour chaque groupe (et donc pour chaque client). Au final, on se retrouve avec une table résultat qui contient une ligne par client, chaque ligne contenant la somme totale que ce client a dépensé dans le magasin.

```
SELECT SUM(montant)
FROM Achats
GROUP BY client ;
```

HAVING

La clause HAVING est similaire à la clause WHERE, à un détail prêt : elle permet d'utiliser des conditions qui impliquent les fonctions SUM, MAX, MIN, AVG et COUNT, ainsi que quelques autres. Elle s'utilise le plus souvent avec un GROUP BY, même si ce n'est pas systématique.

Par exemple, on peut modifier l'exemple précédent de manière à ne conserver que les clients qui ont acheté plus de 500 euros dans le magasin, en utilisant cette requête :

```
SELECT SUM(montant)
FROM Achats
GROUP BY client
HAVING SUM(montant) > 500 ;
```

INSERT, DELETE, UPDATE

Dans cet extrait, nous allons voir comment ajouter, modifier ou supprimer des lignes dans une table. Ces opérations sont respectivement prises en charge par les instructions INSERT, UPDATE et DELETE.

INSERT

INSERT sert à ajouter des lignes dans une table. Sa syntaxe est la suivante :

INSERT INTO nom_table VALUES (liste des valeurs de chaque attribut, séparés par des virgules)

Insertion simple

Prenons la table définie comme suit :

Analyse et Conception d'une base de données

```
create table PERSONNE
(
   ID_PERSONNE int not null ,
   NOM varchar not null ,
   PRENOM varchar not null ,
   AGE int ,
   CATEGORIE_PROFESSIONNELLE varchar ,
   NOMBRE_ENFANTS int ,
   TAILLE float ,

   primary key (ID_PERSONNE) ,
) ;
```

Supposons que nous souhaitons ajouter la personne numéro 50, nommée Pierre Dupont, qui a 25 ans, sans emploi, sans enfants et de taille 174 centimètres. Voici comment utiliser INSERT pour cela :

```
INSERT INTO personne VALUES (50, "Dupont", "Pierre", 25, "Sans emploi", 0, 174)
```

On remarquera que les informations sont données dans l'ordre des colonnes.

Maintenant, passons à l'exemple suivant. Prenons la table suivante :

```
create table ENFANT
(
   ID_ENFANT int not null ,
   NOM varchar not null ,
   PRENOM varchar not null ,
   ADRESSE varchar not null ,
   DATE_NAISSANCE date not null ,
) ;
```

Pour y ajouter l'enfant numéro 27, appelé "Jean Moreno", qui habite "22 rue des tuileries Paris", né le 17/01/2009, voici comment utiliser INSERT :

```
INSERT INTO enfant VALUES (27, "Moreno", "Jean", "22 rue des tuileries Paris", 17/01/2009)
```

Insertion multiple

Il est possible d'insérer plusieurs lignes à la fois dans une table en utilisant une seule instruction INSERT. Pour cela, les diverses lignes à ajouter sont simplement placées les unes après les autres à la suite du VALUES, entre parenthèses.

```
INSERT INTO nom_table VALUES
( première ligne )
( seconde ligne )
( troisième ligne )
( ... )
```

DELETE

Analyse et Conception d'une base de données

L'instruction DELETE permet de supprimer les lignes d'une table. On peut l'utiliser sous deux formes :

- une qui supprime toutes les lignes de la table ;
- une autre qui supprime seulement les lignes qui respectent une certaine condition.

Suppression totale

Pour supprimer toutes les lignes d'une table, il faut préciser quelle est la table concernée. La syntaxe suivante permet de supprimer toutes les lignes de la table nommée "nom_table" :

```
DELETE FROM nom_table ;
```

En supprimant toutes les lignes, la table n'est pas supprimée : on obtient une table vide.

Suppression conditionnelle

Pour supprimer les lignes d'une table qui respectent une condition précise, il faut ajouter quelque chose à la syntaxe précédente, histoire de préciser quelle est la condition en question. Pour préciser la condition, on fait comme avec l'instruction SELECT : on utilise une clause WHERE. La syntaxe suivante permet de supprimer toutes les lignes de la table nommée "nom_table", qui respectent la condition nommée "condition" :

```
DELETE FROM nom_table
WHERE condition ;
```

Prenons l'exemple d'une table "Mineurs" qui mémorise une liste de personnes mineures pour une application judiciaire destinée à un tribunal pour enfants. Cette table mémorise, pour chaque enfant, son nom, prénom, age, date de naissance, et bien d'autres informations dans des attributs éponymes. Tous les ans, cette table est mise à jour pour que l'age mémorisé soit le bon. Cependant, suite à cette mise à jour, des lignes ont un age qui vaut 18, ce qui fait que la ligne correspond à des personnes majeures. Voici la requête qui permet de supprimer ces lignes :

```
DELETE FROM Mineurs WHERE age >= 18 ;
```

UPDATE

Pour mettre à jour certaines lignes d'une table, on doit utiliser l'instruction UPDATE. Celle-ci fonctionne comme pour la suppression des lignes : on doit préciser quelles sont les lignes à modifier avec une condition : toutes les lignes de la table qui respectent cette condition seront modifiées, alors que les autres ne seront pas touchées. Toutes les colonnes sont mises à jour avec la valeur indiquée dans le UPDATE. Pour faire la mise à jour, il faut ainsi préciser :

- quelle table modifier à la suite de l'instruction UPDATE ;
- quelles colonnes modifier et par quoi remplacer leur contenu : c'est le rôle de l'instruction SET ;
- et enfin quelle est la condition avec, encore une fois, un WHERE.

```
UPDATE table
SET colonne_1 = 'valeur 1', colonne_2 = 'valeur 2', colonne_3 = 'valeur 3'
WHERE condition ;
```

Analyse et Conception d'une base de données

Fonction d'agrégation :

- SUM() calculer la somme d'un set de résultat
- MAX() obtenir le résultat maximum (fonctionne bien pour un entier)
- MIN() obtenir le résultat minimum
- COUNT() compter le nombre de lignes dans un résultat
- ROUND() arrondir la valeur
- UPPER() afficher une chaîne en majuscule
- LOWER() afficher une chaîne en minuscule
- NOW() date et heure actuelle
- RAND() retourner un nombre aléatoire
- CONCAT() concaténer des chaînes de caractères
- CURRENT_DATE() date actuelle

SQL DISTINCT

L'utilisation de la commande SELECT en SQL permet de lire toutes les données d'une ou plusieurs colonnes. Cette commande peut potentiellement afficher des lignes en doubles. Pour éviter des redondances dans les résultats il faut simplement ajouter DISTINCT après le mot SELECT.

Commande basique

L'utilisation basique de cette commande consiste alors à effectuer la requête suivante:

```
SELECT DISTINCT ma_colonne
FROM nom_du_tableau
```

Cette requête sélectionne le champ "ma_colonne" de la table "nom_du_tableau" en évitant de retourner des doublons.

Analyse et Conception d’une base de données

SQL LIKE

L’opérateur LIKE est utilisé dans la clause WHERE des requêtes SQL. Ce mot-clé permet d’effectuer une recherche sur un modèle particulier. Il est par exemple possible de rechercher les enregistrements dont la valeur d’une colonne commence par telle ou telle lettre. Les modèles de recherches sont multiple.

Syntaxe

La syntaxe à utiliser pour utiliser l’opérateur LIKE est la suivante :

```
SELECT *
FROM table
WHERE colonne LIKE modele
```

Dans cet exemple le “modèle” n’a pas été défini, mais il ressemble très généralement à l’un des exemples suivants:

- LIKE ‘%a’ : le caractère “%” est un caractère joker qui remplace tous les autres caractères. Ainsi, ce modèle permet de rechercher toutes les chaines de caractère qui se termine par un “a”.
- LIKE ‘a%’ : ce modèle permet de rechercher toutes les lignes de “colonne” qui commence par un “a”.
- LIKE ‘%a%’ : ce modèle est utilisé pour rechercher tous les enregistrement qui utilisent le caractère “a”.
- LIKE ‘pa%on’ : ce modèle permet de rechercher les chaines qui commence par “pa” et qui se terminent par “on”, comme “pantalon” ou “pardon”

Analyse et Conception d’une base de données

- LIKE ‘a_c’ : peu utilisé, le caractère “_” (underscore) peut être remplacé par n’importe quel caractère, mais un seul caractère uniquement (alors que le symbole pourcentage “%” peut être remplacé par un nombre incalculable de caractères . Ainsi, ce modèle permet de retourner les lignes “aac”, “abc” ou même “azc”.

Exemple

Imaginons une table “client” qui contient les enregistrement d’utilisateurs :

id	nom	ville
1	Léon	Lyon
2	Odette	Nice
3	Vivien	Nantes
4	Etienne	Lille

Obtenir les résultats qui commencent par “N”

Si l’ont souhaite obtenir uniquement les clients des villes qui commencent par un “N”, il est possible d’utiliser la requête suivante:

```
SELECT *
FROM client
WHERE ville LIKE 'N%'
```

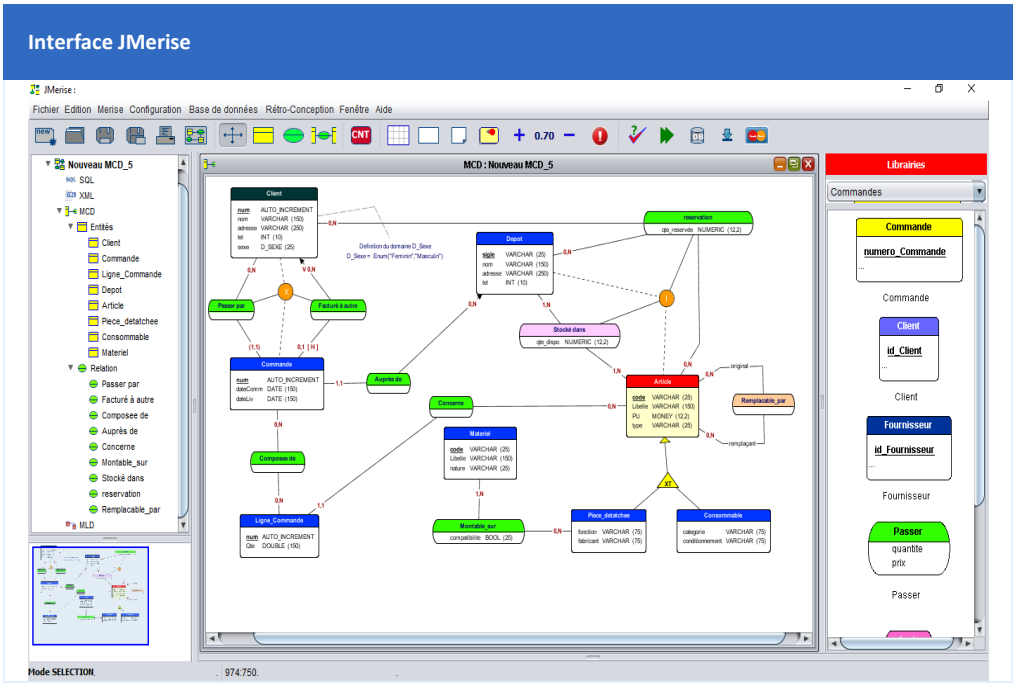
Analyse et Conception d'une base de données

V. Etude Comparative entre MERISE et UML

MERISE est une méthode de modélisation de données et traitements orienté bases de données relationnelles.

Merise a été très utilisée dans les années 1970 et 1980 pour l'informatisation massive des organisations. Cette méthode reste adaptée pour la gestion des projets internes aux organisations, se limitant à un domaine précis. Elle est en revanche moins adaptée aux projets transverses aux organisations, qui gèrent le plus souvent des informations à caractère sociétal (environnemental et social) avec des parties prenantes. Il existe plusieurs logiciels pour son implémentation ex :JMerise.

JMerise : est un logiciel dédié à la modélisation des modèles conceptuels de donnée pour Merise permet les relations réflexives, la généralisation et la spécialisation des entités. Il génère le MLD et le script Mysql..



Le **Langage de Modélisation Unifié (IML)**, de l'anglais *Unified Modeling Language* (**UML**), est un langage de modélisation graphique à base de pictogrammes conçu pour fournir une méthode normalisée pour visualiser la conception d'un système. Il est couramment utilisé en développement logiciel et en conception orientée objet.

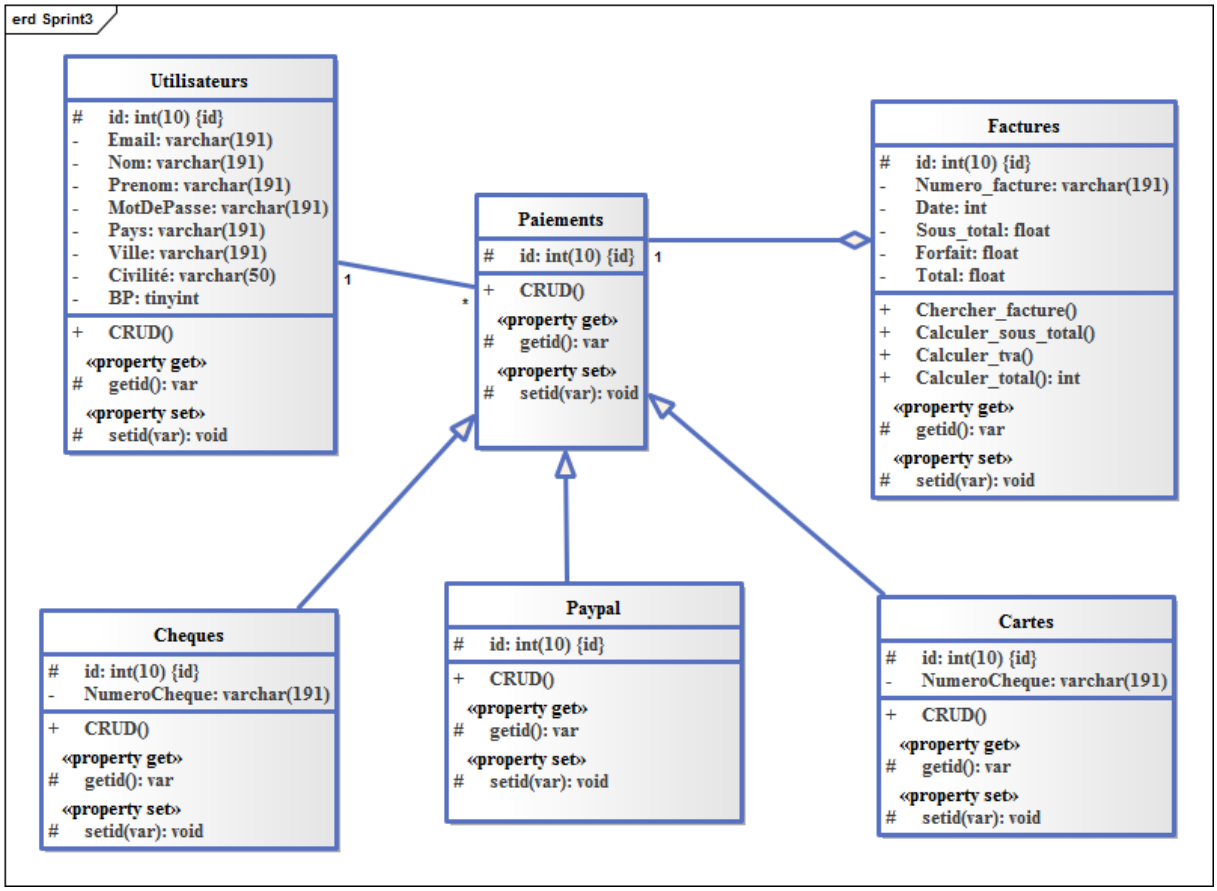
UML est utilisé pour spécifier, visualiser, modifier et construire les documents nécessaires au bon développement d'un logiciel orienté objet. UML offre un standard de modélisation, pour représenter l'architecture logicielle. Les différents éléments représentables sont :

- Activité d'un objet/logiciel
- Acteurs
- Processus
- Schéma de base de données
- Composants logiciels
- Réutilisation de composants

Analyse et Conception d'une base de données

Grâce aux outils de modélisation UML, il est également possible de générer automatiquement tout ou partie du code d'une application logicielle, par exemple en langage Java, à partir des divers documents réalisés.

Exemple de Modélisation avec UML :



Donc on peut en déduire que MERISE est une méthode de modélisation des données alors qu'UML est un langage.

VI. PHPMyAdmin

phpMyAdmin (PMA) est une application Web de gestion pour les systèmes de gestion de base de données MySQL réalisée principalement en PHP et distribuée sous licence GNU GPL.

Il s'agit de l'une des plus célèbres interfaces pour gérer une base de données MySQL sur un serveur PHP. De nombreux hébergeurs, gratuits comme payants, le proposent ce qui évite à l'utilisateur d'avoir à l'installer.

Cette interface pratique permet d'exécuter, très facilement et sans grandes connaissances en bases de données, des requêtes comme les créations de table de données, insertions, mises à jour, suppressions et modifications de structure de la base de données, ainsi que l'attribution et la révocation de droits et l'import/export. Ce système permet de sauvegarder commodément une base de données sous forme de fichier .sql et d'y transférer ses données, même sans connaître SQL.

Analyse et Conception d’une base de données

Les requêtes SQL restent possibles, ce qui permet de les tester interactivement lors de la création d'un site pour les utiliser ensuite en batch (c'est-à-dire en différé) une fois au point.

Exemple d’une interface de phpMyAdmin :