ANALYSE ET CONCEPTION DE BASE DE DONNEES

MERISE,UML ET SQL

I. Notion Analyse et Conception

i. Merise

Merise est une méthode d'analyse et de conception d'un projet informatique. Pour certain, c'est une méthode d'étude et de réalisation informatique pour les systèmes de l'entreprise.

Merise utilise surtout le système informatique de l'entreprise.

Le système informatique est un ensemble organise de ressources qui permettent de collecter, stocker, traiter et distribuer de l'information grâce à un ordinateur.

Merise est une nouvelle approche des systèmes informatiques qui permet d'implémenter des bases de données relationnelles. En outre dans sa démarche, il assure une séparation de l'étude des données et des traitements. Ainsi il existe plusieurs niveaux de conceptions.

	Données	**Traitements**
Niveau conceptuel	MCD (Modèle Conceptuel des Données)	MCT (Modèle Conceptuel des Traitements)
Niveau organisationnel et logique	MLD (Modèle Logique des Données)	MOT (Modèle Organisationnel des Traitements)
Niveau opérationnel et physique.	MPD (Modèle Physique des données)	MOPT (Modèle opérationnel des Traitements)

ii. UML

UML se définit comme un langage de modélisation graphique et textuel destiné à comprendre et décrire des besoins, spécifier et documenter des systèmes, esquisser des architectures logicielles, concevoir des solutions et communiquer des points de vue.

UML unifie à la fois les notations et les concepts orientés objet. Il ne s'agit pas d'une simple notation graphique, car les concepts transmis par un diagramme ont une sémantique précise et sont porteurs de sens au même titre que les mots d'un langage.

UML unifie également les notations nécessaires aux différentes activités d'un processus de développement et offre, par ce biais, le moyen d'établir le suivi des décisions prises, depuis l'expression de besoin jusqu'au codage. Dans ce cadre, un concept appartenant aux exigences des utilisateurs projette sa réalité dans le modèle de conception et dans le codage. Le fil tendu entre les différentes étapes de construction permet alors de remonter du code aux besoins et d'en comprendre les tenants et les aboutissants. En d'autres termes, on peut retrouver la nécessité d'un bloc de code en se référant à son origine dans le modèle des besoins.

UML 2 s'articule autour de treize types de diagrammes, chacun d'eux étant dédié à la représentation des concepts particuliers d'un système logiciel. Ces types de diagrammes sont répartis en deux grands groupes :

- **Six diagrammes structurels :**

- **Diagramme de classes** : Il montre les briques de base statiques : classes, associations, interfaces, attributs, opérations, généralisations, etc.
- **Diagramme d'objets** : Il montre les instances des éléments structurels et leurs liens à l'exécution.
- **Diagramme de packages** : Il montre l'organisation logique du modèle et les relations entre packages.
- **Diagramme de structure composite** : Il montre l'organisation interne d'un élément statique complexe.
- **Diagramme de composants** : Il montre des structures complexes, avec leurs interfaces fournies et requises.
- **Diagramme de déploiement** : Il montre le déploiement physique des « artefacts » sur les ressources matérielles.

- **Sept diagrammes comportementaux :**

- **Diagramme de cas d'utilisation** : Il montre les interactions fonctionnelles entre les acteurs et le système à l'étude.
- **Diagramme de vue d'ensemble des interactions** : Il fusionne les diagrammes d'activité et de séquence pour combiner des fragments d'interaction avec des décisions et des flots.
- **Diagramme de séquence** : Il montre la séquence verticale des messages passés entre objets au sein d'une interaction.
- **Diagramme de communication** : Il montre la communication entre objets dans le plan au sein d'une interaction.
- **Diagramme de temps** : Il fusionne les diagrammes d'états et de séquence pour montrer l'évolution de l'état d'un objet au cours du temps
- **Diagramme d'activité** : Il montre l'enchaînement des actions et décisions au sein d'une activité.
- **Diagramme d'états** : Il montre les différents états et transitions possibles des objets d'une classe.

L'ensemble des treize types de diagrammes UML peut ainsi être résumé sur la figure suivante :

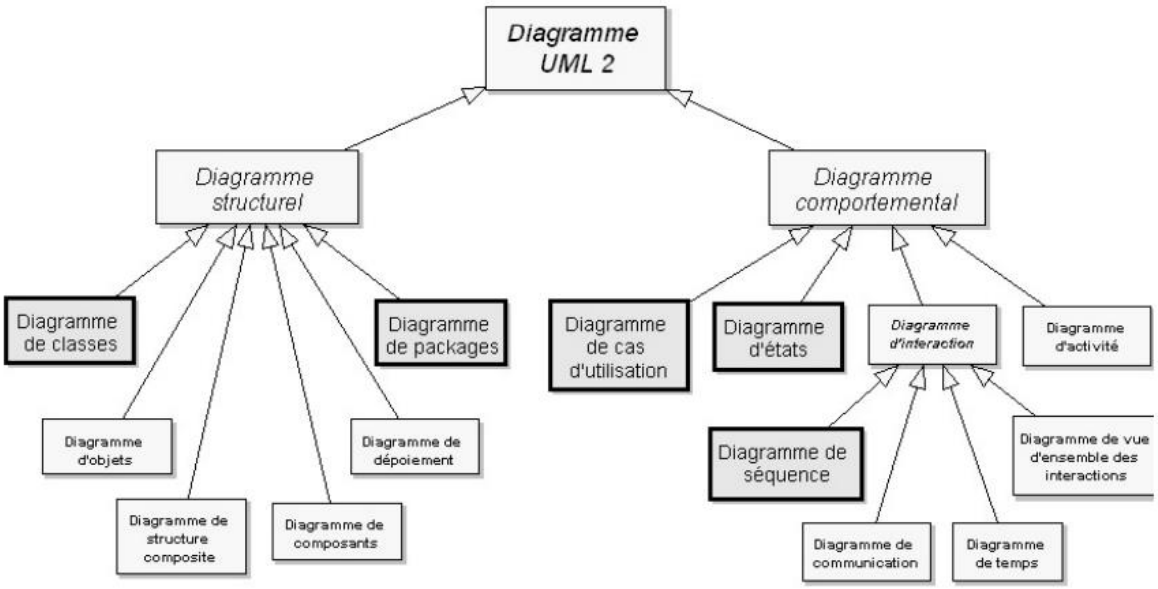


Figure 1–12 Les diagrammes UML utilisés dans notre démarche agile

iii. Etude Comparative

a) Niveaux d'abstraction

L'approche Merise : Le cycle d'abstraction permet de sérier les niveaux de préoccupations lors de la description ou de l'analyse du système. Les trois niveaux retenus correspondent à des degrés de stabilité et d'invariance de moins en moins élevés.

- Le niveau conceptuel ;
- Le niveau logique ;
- Le niveau physique.

L'approche UML propose différentes notions (cas d'utilisation, paquetage, classe, composant, nœud) et différents diagrammes pour modéliser le système aux différents niveaux d'abstraction.

b) Approche fonctionnelle

L'approche Merise propose une approche descendante où le système réel est décomposé en activités, elles-mêmes déclinées en fonctions. Les fonctions sont composées de règles de gestion, elles-mêmes regroupées en opérations. Ces règles de gestion au niveau conceptuel génèrent des modules décomposés en modules plus simples et ainsi de suite jusqu'à obtenir des modules élémentaires... Les limites d'une telle approche résident dans le fait que les modules sont difficilement extensibles et exploitables pour de nouveaux systèmes.

L'approche UML : Les fonctions cèdent la place aux cas d'utilisation qui permettent de situer les besoins de l'utilisateur dans le contexte réel. A chaque scénario correspond des diagrammes d'interaction entre les objets du système et non pas un diagramme de fonction...

c) Dualité des données -traitements

L'approche Merise propose de considérer le système réel selon deux points de vue : un point de vue statique (les données), un point de vue dynamique (les traitements). Il s'agit d'avoir une vision duale du système réel pour bénéficier de l'impression de relief qui en résulte, et donc consolider et valider le système final.

L'approche UML : L'approche objet associe les informations et les traitements. De cette façon, elle assure un certain niveau de cohérence.

II. Concepts Analyse et Conception

A) MCD

Le modèle conceptuel des données (MCD) a pour but de représenter de façon structurées les données qui seront utilisées par le système d'information. Le modèle conceptuel des données décrit la sémantique c'est à-dire le sens attaché à ces données et à leurs rapports et non à l'utilisation qui peut en être faite.

- Les entités

  Une entité est une population d'individus homogènes. Par exemple, les produits ou les articles vendus par une entreprise peuvent être regroupés dans une même entité **articles** (figure 1), car d'un article à l'autre, les informations ne changent pas de nature (à chaque fois, il s'agit de la désignation, le prix unitaire, etc.)

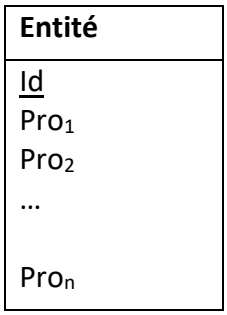
- Les attributs
  Un attribut est une propriété d'une entité ou d'une association Dans notre exemple, le prix unitaire est un attribut de l'entité articles, nom est un attribut de l'entité clients
- Les Occurrences
  Une occurrence d'un objet est un élément individualise de cet objet. Elle a une valeur unique pour chaque propriété de cet objet.
  Exemple :
  Client

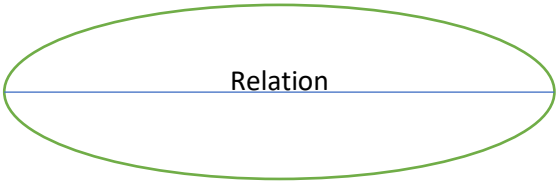
Prénom	Nom	Adresse	Téléphone	Date Naissance
Adama	Diop	23 Rue	76XXXXXXX	26-10-1996
Marieme	Sidibé	___	___	29-01-1984

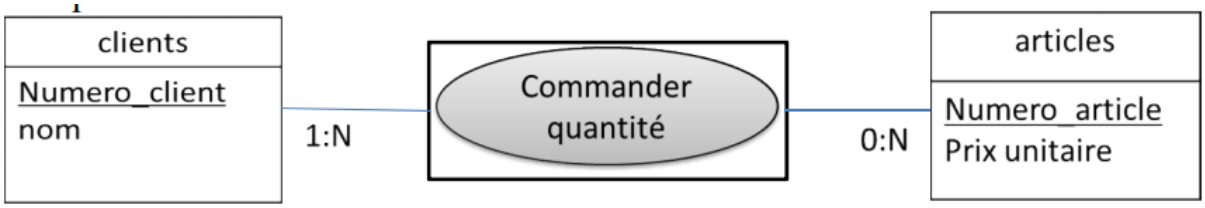
  L'individu Adama Diop est une occurrence de l'objet Client
- Les Cardinalités
  La cardinalité dans une relation donnée est le nombre d'occurrence d'un objet qui entre dans la relation.
  Les cardinalités se traduisent par deux nombres :

- $N_1\{0, 1\}$ qui indique le nombre minimum d'occurrence d'un objet qui entre dans la relation
- $N_2\{0, N\}$ qui indique le nombre maximum d'occurrence qui entre dans la relation.

- Formalisme de Représentation







B) MLD

Maintenant que le MCD est établi, on peut le traduire en différents systèmes logiques et notamment les bases de données relationnelles qui proposent une vision plus concrète pour modéliser la situation

a) MLDR
b) Modèle Relationnelle
   1) Tables, lignes et colonnes

Lorsque des données ont la même structure (comme par exemple, les renseignements relatifs aux clients), on peut les organiser en table dans laquelle les colonnes décrivent les champs en commun et les lignes contiennent les valeurs de ces champs pour chaque enregistrement.

Numéro client	nom	prénom	adresse
1	Diop	Paul	10, rue Grd Dakar
2	Sall	Jean	22, rue Médina
3	Diallo	Mamadou	Fass
4	Ka	Ousmane	Hlm 3
…	…	…	…

c) Règles de passage du MCD au MLDR

Pour traduire un MCD en un MLDR, il suffit d'appliquer cinq règles :

Notations : on dit qu'une association binaire (entre deux entités ou réflexive) est de type :

- 1 : 1 (un à un) si aucune des deux cardinalités maximales n'est n ;
- 1 : n (un à plusieurs) si une des deux cardinalités maximales est n ;

- n : m (plusieurs à plusieurs) si les deux cardinalités maximales sont n.

En fait, un schéma relationnel ne peut faire la différence entre 0,n et 1,n. Par contre, il peut la faire entre 0,1 et 1,1 (règles 2 et 4).

Règle 1 : toute entité devient une table dans laquelle les attributs deviennent les colonnes. L'identifiant de l'entité constitue alors la clé primaire de la table.

Règle 2 : une association binaire de type 1 : n disparaît, au profit d'une clé étrangère dans la table, côté 0,1 ou 1,1 qui référence la clé primaire de l'autre table. Cette clé étrangère ne peut pas recevoir la valeur vide si la cardinalité est 1,1.

Règle 3 : une association binaire de type n : m devient une table supplémentaire (parfois appelé table de jonction, table de jointure ou table d'association) dont la clé primaire est composée de deux clés étrangères (qui référencent les deux clés primaires des deux tables en association). Les attributs de l'association deviennent des colonnes de cette nouvelle table.

Règle 4 : une association binaire de type 1 : 1 est traduite comme une association binaire de type 1 : n sauf que la clé étrangère se voit imposer une contrainte d'unicité en plus d'une éventuelle contrainte de non vacuité (cette contrainte d'unicité impose à la colonne correspondante de ne prendre que des valeurs distinctes). Si les associations fantômes ont été éliminées, il devrait y avoir au moins un côté de cardinalité 0,1. C'est alors dans la table du côté opposé que doit aller la clé étrangère. Si les deux côtés sont de cardinalité 0,1 alors la clé étrangère peut être placé indifféremment dans l'une des deux tables.

Règle 5 : une association non binaire est traduite par une table supplémentaire dont la clé primaire est composée d'autant de clés étrangères que d'entités en association. Les attributs de l'association deviennent des colonnes de cette nouvelle table.

d) Formalisme de représentation

III. SQL

Pour communiquer avec une base de données, il faut lui envoyer des commandes ou des instructions appelées requêtes. Que ce soit pour la création, la suppression d'une table, la modification, l'insertion ou la sélection de données, le langage standard de requêtes est SQL (Standard Query Language).

- SQL est un langage de définition de données (LDD, ou DDL (en anglais), Data definition language) : il permet de modifier la structure d'une base de données relationnelle ;
- SQL est un langage de manipulation de données (LMD ou DML (en anglais), Data manipulation language) : il permet de consulter, de supprimer et d'effectuer des mises à jour sur le contenu d'une base de données relationnelle ;
- SQL un langage de contrôle de transactions (LCT, ou en anglais TCL, Transaction control language) : il permet de gérer les transactions, c'est-à-dire rendre atomique divers ordres enchaînés en séquence ;

- SQL intègre aussi d'autres modules destinés notamment à écrire des routines (procédures, fonctions ou déclencheurs) et interagir avec des langages externes.

a) Langage de définition des données

1) CREATE

CREATE TABLE Nom_de_table

(Nom_attribut1 TYPE [NOT NULL]

[, Nom_attribut2 TYPE [NOT NULL]]

.....

[, Nom_attributn TYPE [NOT NULL]]

[CONSTRAINT nomcontrainte1 Typecontrainte1]

[CONSTRAINT nomcontrainte2 Typecontrainte2]

…..

);

Types de données :

CHAR (n)**, VARCHAR** (n)**, NUMBER** (n), **DECIMAL** (M, D), **DATE, ETC.**

Définition des contraintes d 'intégrité :

CONSTRAINT nomcontrainte **PRIMARY KEY** (Nom_attribut1[, Nom_attribut2] ...)

CONSTRAINT nomcontrainte **FOREIGN KEY** (Nom_attribut1[, Nom_attribut2] ...)

REFERENCES Nom_table [(nom_attribut)]

[**ON DELETE** { CASCADE | SET NULL }]

[**ON UPDATE** CASCADE]

CONSTRAINT nomcontrainte **CHECK** (Condition)

CONSTRAINT nomcontrainte **UNIQUE** (Nom_attribut1[, Nom_attribut2] ...)

2) DROP

Suppression d 'objet :

DROP TABLE Nom_table;

DROP INDEX Nom_Index**;**

DROP VIEW Nom_Vue**;**

3) ALTER

Modification de tables :

Ajout, modification, suppression de colonnes ou de contraintes :

ALTER TABLE Nomtable **ADD** (Nomattribut TYPE [NOT NULL] [,….]);

ALTER TABLE Nomtable **ADD** [**CONSTRAINT** Nomcontrainte] typecontrainte;

ALTER TABLE Nomtable **DROP CONSTRAINT** Nomcontrainte [CASCADE];

ALTER TABLE Nom_table1

ADD FOREIGN KEY (Nom_attr1 [, Nom_attr2]..) **REFERENCES** Nom_table2;

ALTER TABLE Nom_table1

ADD PRIMARY KEY (Nom_attr1 [, Nom_attr2]..);

b) Langage de manipulation de données (LMD)

1) INSERT

Insertion de lignes dans une table :

INSERT INTO Nom_table [(Nom1, ..Nomn)]{**VALUES** (valeur1,...valeurn) | requête};

2) UPDATE

Modification de valeurs contenues dans des lignes :

UPDATE Nom_table **SET** col1=expr1 [,col2=expr2] .... [WHERE condition];

UPDATE Nom_table **SET** (col1,col2,....)= (requête) [WHERE condition];

3) DELETE

Suppression de lignes :

DELETE FROM Nom_table [WHERE condition];

c) Langage d'interrogation de données (LID)

1) JOINTURE

Jointure :

SELECT *

FROM Nom_de_relation1, Nom_de_relation2

WHERE Condition_de_jointure;

Exemple :

SELECT ***** FROM CATEGORIE **C**, PRODUIT **P**

WHERE **C**.CODCAT=**P**.CODCAT;

Jointure SQL2:

SELECT *

FROM Nom_de_relation1 INNER JOIN Nom_de_relation2 **ON**

Condition_de_jointure;

2) PROJECTION

Projection :

SELECT DISTINCT liste_des_attributs_de projection

FROM Nom_Relation**;**

Exemple:

SELECT DISTINCT VILLE FROM PERSONNE;

3) SOUS REQUETE

SQL permet l'imbrication de sous requêtes au niveau de la clause WHERE d'où le terme "structuré" dans Structured Query Language.

Les sous-requêtes sont utilisées :

- dans des prédicats de comparaison (=, <>, <=, >, >=)
- dans des prédicats IN
- dans des prédicats EXISTS
- dans des prédicats ALL ou ANY

Une sous-requête dans un prédicat de comparaison doit se réduire à une seule valeur ("singleton select")

- Une sous-requête dans un prédicat IN, ALL ou ANY, doit représenter une table à colonne unique
- L'utilisation de constructions du type "IN sous-requête" permet d'exprimer

Jointure procédurale:

SELECT *

FROM Nom_de_relation1

WHERE Nom_Attribut1 **IN**

(SELECT Nom_attribut2

FROM Relation2);

Sous interrogation dans la clause FROM:

SELECT *

FROM Nom_relation1, (SELECT…FROM Nom_relation2) alias_rel2

WHERE (condition);

Sous interrogations synchronisées:

SELECT *

FROM Nom_relation1 alias_rel1

WHERE Nom_col θ **(SELECT …**

FROM Nom_relation2 alias_rel2

WHERE alias_rel1.x θ alias rel2.y)

[AND…..];

θ à prendre parmi { <, >, <=, >=, IN }

4) ALIAS

Synonyme de nom de table (ou alias) :

- On peut introduire dans la clause FROM un synonyme (alias) à un nom de table en le plaçant immédiatement après le nom de la table
- Les noms de table ou les synonymes peuvent être utilisés pour préfixer les noms de colonnes dans le SELECT
- Les préfixes ne sont obligatoires que dans des cas particuliers (par ex. pour une auto-jointure) ; leur emploi est cependant conseillé pour la clarté Synonyme de nom de table (ou alias)

Un alias est utilisé par SQL comme une variable de parcours de table (dite variable de corrélation) désignant à tout instant une ligne de la table

5) COMMANDE

a) UNION

SELECT Liste_attributs

FROM Nom_De_Relation1

[WHERE <Condition de restriction> **]**

UNION

SELECT Liste_attributs

FROM Nom_De_Relation2

[WHERE <Condition de restriction>]

b) GROUP BY

SELECT Liste_attributs **FROM** Nom_De_Relation

GROUP BY Liste_attributs_de_regroupement

Exemple :

SELECT CODCAT, Count(*) FROM PRODUIT WHERE PRIX > 100 GROUP BY CODCAT;

SELECT Liste_attributs **FROM** Nom_De_Relation

GROUP BY Liste_attributs_de_regroupement

HAVING Condition

Exemple :

SELECT CODCAT, Count(*) FROM PRODUIT WHERE PRIX > 100

GROUP BY CODCAT HAVING Count (*) > 20;

c) ORDER BY

La clause ORDER BY permet de spécifier les colonnes définissant les critères de tri. Le tri se fera d'abord selon la première colonne spécifiée, puis selon la deuxième colonne etc...

Exemple :

SELECT * FROM produit ORDER BY marque, prix DESC L'ordre de tri est précisé par ASC (croissant) ou DESC (décroissant) ;

par défaut ASC

d) HAVING

La clause HAVING permet de spécifier une condition de restriction des groupes.

Elle sert à éliminer certains groupes, comme WHERE sert à éliminer des lignes

Exemple :

SELECT P.marque, AVG (P.prix) FROM produit P GROUP BY P.marque HAVING AVG (P.prix) < 5000;

6) FONCTION

a) COUNT

Comptage : COUNT([DISTINCT|ALL]{*|expr})

b) LIMIT

Limiter le nombre de ligne à afficher :

Exemple : Afficher 5 fournisseurs à partir du dixième.

SELECT * FROM fournisseurs LIMIT 9,5 ;

Exemple : Afficher les 3 premiers fournisseurs de la table.

SELET * FROM fournisseurs LIMIT 3 ;

c) LIKE

SELECT * FROM client C WHERE C.nom LIKE '_A%' ;

Le prédicat LIKE compare une chaîne avec un modèle (_) remplace n'importe quel caractère (%) remplace n'importe quelle suite de caractères

d) BETWEEN

SELECT * FROM produit WHERE prix BETWEEN 5000 AND 12000 ;

Le prédicat BETWEEN teste l'appartenance à un intervalle

e) AVG

Moyenne : AVG([DISTINCT|ALL] expr)

f) MIN

Minimum : MIN([DISTINCT|ALL] expr)

g) MAX

Maximum : MAX([DISTINCT|ALL] expr)

h) SUM

Somme : SUM([DISTINCT|ALL] expr)

Exemple :

```
SELECT COUNT(*) FROM PRODUIT;
SELECT MAX(PRIX_HT*TAUX_TVA)
        FROM PRODUIT P, CATEGORIE C
        WHERE P.CODCAT= C.CODCAT;
```