Analyse et Conception de Base de Données

Historique

• Première génération (années 60)

- Séparation de la description des données et des programmes d'application
- Traitement de fichiers reliés par des structures de graphe
- SGBD IDMS (Integrated Database Management System)
- Deuxième génération (années 70)

• Simplification du SGBD externe

- • Modèle relationnel
- • Langage non procédural
- • SGBD ORACLE, INFORMIX, …

• Troisième génération (années 80)

- • SGBD objet
  - • SGBD Relationnel objet : ORACLE 8 à 12
  - • SGBD orienté objet : O2, Versant

• Quatrième génération (années 90)

- • Bases de données et internet
  - • Entrepôts de données (data Warehouse)
  - • Fouille de données (data mining)
- • Actuellement: Big Data, Data Lake

Base de données en général

• A quoi servent les bases de données?

- Stockage des informations :
  - sur un support informatique
  - pendant une longue période
  - de taille importante
  - accès multi-utilisateurs
- Il faut donc :
  - • gérer de manière efficace les accès aux disques
  - • proposer une définition structurée des données
  - • éviter les redondances

I. Notion Analyse et Conception

● Merise

C’est une méthode d'analyse et de conception des systèmes d'information basée sur le principe de la séparation des données et des traitements. Elle possède plusieurs modèles qui sont répartis sur 3 niveaux (Le niveau conceptuel, le niveau logique ou organisationnel, le niveau physique).

Les questions abordées à chaque niveau

Niveaux	Questions	Exemples
CONCEPTUEL	GESTION, « METIER »	données traitées, règles de gestion, enchaînements des traitements…
ORGANISATIONNEL LOGIQUE	ORGANISATION	partage homme/machine, interactif/différé, organisation des données et traitements, distribution…
PHYSIQUE	TECHNIQUE	programmes, écrans, états, organisation physique des données, matériel, réseau…

● UML

UML, c’est l’acronyme anglais pour « Unified Modeling Language ». On le traduit par « Langage de modélisation unifié ». La notation UML est un **langage visuel** constitué d’un ensemble de schémas, appelés des **diagrammes**, qui donnent chacun une vision différente du projet à traiter. UML nous fournit donc des diagrammes pour **représenter** le logiciel à développer : son fonctionnement, sa mise en route, les actions susceptibles d’être effectuées par le logiciel, etc.

● Etude Comparative

Merise ou UML ?

Les « méthodologues » disent qu'une méthode, pour être opérationnelle, doit avoir 3 composantes :

Une démarche (les étapes, phases et tâches de mise en oeuvre) ;

Des formalismes (les modélisations et les techniques de transformations) ;

Une organisation et des moyens de mise en œuvre.

Merise s'est attachée, en son temps, à proposer un ensemble « cohérent » sur ces trois composantes. Certaines ont vieilli et ont dû être réactualisées (la démarche), d'autre « tienne encore le chemin » (la modélisation).

UML se positionne exclusivement comme un ensemble de formalismes. Il faut y associer une démarche et une organisation pour constituer une méthode.

Merise se positionne comme une méthode de conception de système d'information organisationnel, plus tournée vers la compréhension et la formalisation des besoins du métier que vers la réalisation de logiciel. En sens, Merise se réclame plus de l'ingénierie du système d'information que du génie logiciel. Jamais Merise ne s'est voulu une méthode de développement de logiciel ni de programmation.

UML, de par son origine (la programmation objet) s'affirme comme un ensemble de formalismes pour la conception de logiciel à base de langage objet.

Merise est encore tout à fait valable pour :

La modélisation des données en vue de la construction d'une base de données relationnelles, la modélisation des processus métiers d'un système d'information automatisé en partie par du logiciel,

La formalisation des besoins utilisateur dans la cadre de cahier des charges utilisateur, en vue de la conception d'un logiciel adapté.

UML est idéal pour:

Concevoir et déployer une architecture logiciel développée dans un langage orienté objet (Java, C++, VB.Net,...).

Pour modéliser les données (le modèle de classe réduit sans méthodes et stéréotypé en entités), mais avec des lacunes que ne présentait pas l'entité relation de Merise.

Pour modéliser le fonctionnement métier (le diagramme d'activité et de cas d'utilisation) qui sont des formalismes très anciens qu'avait, en son temps, amélioré Merise...

II. Concepts Analyse et Conception

A. MCD : Modèles Conceptuelles de données

- **Les entités :** Une entité est ensemble d'éléments de même nature. Une entité est la représentation d'un élément matériel ou immatériel ayant un rôle dans le système que l'on désire décrire
- ***Les attributs :*** Les attributs d'entité sont les éléments d'information de base attachés à une entité.

- ***Les Occurrences :*** En base de données, on distingue le modèle d'une côté (la structure avec les entités et les relations) et d'un autre côté le contenu (il faut bien les remplir avec des valeurs ces beaux modèles).

  Une **occurrence**, c'est tout simplement une **« ligne » de valeurs**. Dans une entité, une occurrence correspond à l'ensemble des valeurs des propriétés rattachées à un seul identifiant. Dans une relation, une occurrence correspond à l'ensemble des valeurs des propriétés de la relation (représenté par les clés de chaque entité liée) : on l'appelle alors une **occurrence de relation**

- ***Les Cardinalités***

Le MCD est l'élément le plus connu de MERISE et certainement le plus utile. Il permet d'établir une représentation claire des données du SI et définit les dépendances fonctionnelles de ces données entre elles.
Les éléments utilisés pour la formalisation d'un MCD sont les suivants :

Entité Type	Définition d'entités (objets physiques ou abstraits) ayant des caractéristiques comparables.
Relation Type	Définition d'une Association liant plusieurs Entités Types. Signification d'un lien entre deux ou plusieurs types d'objets.
Propriété Type	Définition d'une caractéristique d'un objet ou d'une association. Une propriété Type est elle-même caractérisé par un type (Chiffre ou Texte ...) et une longueur. L'ensemble des propriétés types du MCD compose le dictionnaire des données.
Identifiant	Propriété Type ou concaténation de Propriétés Types permettant de distinguer une entité parmi toute les autres dans une Entité Type.
Cardinalité minimum	Nombre minimum de fois où une entité est concernée par l'association. 0 indique que les entités ne sont pas obligatoirement concernés par l'association.
Cardinalité maximum	Nombre maximum de fois où une entité est concernée par l'association. N signifie plusieurs fois sans préciser de nombre. Ce nombre ne peut être égal à 0.

- ***Formalisme de Représentation***

Règles à suivre pour l'établissement d'un MCD

Normalisation

1ere forme Normale	Chaque entité doit disposer d'un identifiant qui la caractérise de manière unique. Le numéro de licence est unique pour chaque PILOTE
2eme forme ormale	Les propriétés d'une entité ne doivent dépendre que de l'identifiant de l'entité et non d'une partie de cet identifiant. Un identifiant peut être composé de la concaténation de plusieurs propriétés.
3eme forme Normale	Les propriétés d'une entité doivent dépendre de l'identifiant de l'entité de manière directe. Exemple : Si l'on rajoutait dans l'entité PILOTE une propriété " Description du Niveau " cette normalisation ne serait pas respectée car la Description du Niveau est directement dépendante du Niveau et non du " numéro de licence ".
Forme Normale de BOYCE-CODD	Pour les identifiants composés de plusieurs propriétés, ces dernières ne doivent pas être dépendantes d'une autre propriété de l'entité.
Normalisation des relations	Les propriétés des relations doivent dépendre de tous les identifiants des entités associées.
Décomposition des relations	Les relations dont le nombre d'entités associé est trop important (supérieur à 3) doivent être décomposées en plusieurs relations. Cette décomposition ne peut se faire qu'à la condition d'avoir une cardinalité minimum égale à 1.

B. MLD : Modèle Logique de données

Le modèle logique de données est une représentation du MCD valide en fonction du matériel et logique utilisés dans l'automatisation. Le modèle logique est le modèle conceptuel plus la réponse aux contraintes d'organisation des données .en autre, il nous permet de préfigurer le temps d'accès et l'espace nécessaire.

- ***MLDR (Modèle Logique de données relationnelles)***

1 - Individu et attributs

Chaque objet du système d'information peut être défini comme une relation entre l'individu et les attributs qui le décrivent.

Exemple : pour l'objet BUS, il y a une relation entre l'individu BUS et les attributs numéro, modèle, motorisation, nombre de places, date de mise en service.

2 - Atomicité des attributs

Chaque attribut est le plus élémentaire possible : c'est l'atomicité.

Exemple : une donnée adresse 8 rue Foch 38000 Grenoble peut être décomposée en trois attributs élémentaires :

adresse rue 8 rue Foch

code postal 38000

ville Grenoble

3 - Clé primaire

Un attribut permet d'identifier sans ambiguïté un individu parmi les autres : c'est la clé primaire.

Exemple : pour le BUS, c'est manifestement le numéro qui permet de repérer sans ambiguïté un bus parmi les autres bus de la compagnie.

La clé primaire obéit à trois règles :

unicité : la clé primaire permet d'accéder à un individu unique, comme une clé permet d'ouvrir une porte. Deux individus ne pourront avoir la même clé primaire sinon, on les confondrait (si une clé permet d'ouvrir plusieurs portes, on ne pourra pas savoir quelle porte sera ouverte à la simple vue de la clé...) ;

existence : tout individu doit pouvoir se voir affecter une valeur de clé primaire (une valeur de clé primaire ne doit pas être vide) ;

stabilité : la valeur de la clé primaire d'un individu ne doit pas pouvoir changer (exemple dans le tableau du point 5: le bus GX317 mis en service le 01/03/2002 gardera toujours le numéro 25).

De plus, la clé primaire ne doit pas être empruntée à une autre organisation (ici, le numéro d'immatriculation automobile attribué par la Préfecture ne conviendrait donc pas).

4 - Présentation des relations

La relation BUS sera représentée comme suit :

BUS (Numéro bus, Modèle bus, Motorisation bus, Nb_places_bus, Date_mise_serv_bus)

La clé primaire est soulignée. La relation LIGNE sera représentée comme suit :

LIGNE (Numéro-ligne, Départ ligne, Terminus-ligne)

Afin que chaque attribut soit repéré par son nom sans aucune hésitation, on distingue ici les deux numéros pour ne pas les confondre : Numéro_bus d'une part, Numéro_ligne d'autre part. Pour mieux repérer les attributs, on peut faire suivre le nom strict de l'attribut par le nom (ou le début du nom) de la relation (comme ici avec bus ou ligne).

Pour faciliter leur manipulation, certains attributs peuvent se voir affecter un nom symbolique (Nb_places_bus, Date_mise_serv_bus).

5 - Tuples de la relation (occurrence)

Il est possible de visualiser différents exemples d'une relation. Ainsi, de même qu'il y a différents bus, la relation BUS prendra des valeurs pour chacun de ses attributs.

Une ligne de la relation = un individu = un tuple = une occurrence

Exemple: le bus 21 est un tuple de la relation BUS.

6 - Valeurs des attributs

Chacun des attributs prend une valeur pour un individu (tuple) donné (le bus 21 n'a qu'une seule valeur de Numéro_bus - 21 -, une seule valeur de Modèle_bus - GX317 -, une seule valeur de Motorisation_bus Diesel-, une seule valeur de Nb_places_bus - 107 -, une seule valeur de Date_mise_serv_bus - 12/10/2000).

L'ensemble des valeurs que peut prendre un attribut s'appelle le domaine de valeurs : ce domaine peut être un ensemble de valeurs, un intervalle (de 1 à 100) ou défini par rapport à la taille (e.g. entier -> nombres décimaux non possibles).

Exemple : Motorisation_bus ne peut prendre que Diesel ou Gaz comme valeur : le domaine de valeurs est donc {Diesel, Gaz} ; si on affecte à Motorisation_bus la valeur Essence, on ne respecte pas la contrainte d'intégrité de domaine, Essence ne figurant pas dans le domaine de valeurs de l'attribut.

7 - Dépendance fonctionnelle

La connaissance d'une valeur de la clé primaire permet de connaître la valeur de chacun des attributs de l'individu : c'est une dépendance fonctionnelle.

Exemple : connaître le numéro 25 permettra de connaître le modèle GX317, la motorisation Gaz, le nombre de places 82 et la date de mise en service 01/03/2002,

Il ne peut y avoir d'autre valeur pour chacun des attributs. La dépendance fonctionnelle peut être schématisée par une flèche.

Exemple : Numéro_bus -> Modèle_bus signifie que Modèle_bus dépend de Numéro_bus (ou que la valeur de Numéro_bus détermine celle de Modèle_bus) : la connaissance d'une valeur de Numéro_bus entraîne la connaissance d'une valeur unique du Modèle_bus.

Pour la relation BUS, on pourra écrire l'ensemble des dépendances fonctionnelles suivantes

Numéro_bus -> Modèle_bus

Numéro_bus -> Motorisation_bus

Numéro_bus -> Nb_places_bus

Numéro_bus -> Date_mise_serv_bus

Les dépendances fonctionnelles permettent de construire les relations en déterminant de façon certaine la clé primaire : une relation doit n'avoir que des dépendances fonctionnelles simples et directes entre la clé primaire et les autres attributs.

Une clé primaire composée de plusieurs attributs peut être nécessaire pour obtenir, de manière unique, la valeur d'un autre attribut (cf 8.2).

8.1 - Lien entre les relations, Cas simple

Il existe une interdépendance entre les deux objets BUS et LIGNE: les 150 bus desservent 17 lignes, chaque bus desservant une seule ligne. Comment la représenter dans la base de données ?

Si je me place du côté de la ligne, je peux avoir plusieurs bus (150 bus desservent 17 lignes) la représentation nous obligerait à ajouter plusieurs numéros de bus (mais combien ?) dans la relation LIGNE.

Si, par contre, je me place dans la relation BUS, il suffit d'ajouter un attribut qui permettrait de voir à quelle ligne est affecté le bus : chaque bus dessert une seule ligne.

Quel attribut est le mieux placé pour montrer sans ambiguïté la ligne ? La clé primaire de la relation LIGNE, ici le Numéro ligne.

Cela conduit au schéma relationnel complet:

BUS (Numéro_bus, Modèle bus, Motorisation_bus, Nb_places_bus, Date_mise_serv_bus, #Numéro-ligne) LIGNE (Numéro_ligne, Dépard_ligne, Terminus_ligne)

L'attribut ajouté qui « représente » la clé primaire s'appelle clé étrangère.

Pour ne pas confondre clé primaire et clé étrangère, les attributs clés étrangères des relations peuvent être préfixés d'un #.

8.2 - Lien entre les relations, Cas complexe

Supposons que l'affection des bus aux lignes est la suivante: les 150 bus desservent 17 lignes, chaque bus pouvant desservir plusieurs lignes.

Extrait des affectations: ligne 1 : bus n° 2, 14, 15, 16, 24 ; ligne 2 : bus n° 3, 11, 15, 18 ; ligne 3 : bus n° 1, 5, 11

Puisqu'une ligne se voit affecter plusieurs bus, et que chaque bus peut desservir plusieurs lignes (dans cet exemple), il est difficile de trouver une représentation qui évite la répétition des attributs.

Dans ce cas, on ajoute une relation (Desservir) pour apparier les bus et les lignes :

BUS (Numéro_bus, Modèle_bus, Motorisation_bus, Nb_places_bus, Date_mise_serv_bus)

LIGNE (Numéro_ligne, Dépard_ligne, Terminus_ligne)

DESSERVIR (#Numéro_bus, #Numéro_ligne)

Pour définir la clé primaire de la relation DESSERVIR, on utilise la clé primaire des deux autres relations.

Bien entendu, comme pour n'importe quelle relation, cette relation peut avoir d'autres attributs.

9 - Présentation du schéma relationnel

Pour une meilleure visualisation du lien entre les relations, le schéma relationnel présenté en ligne peut être visualisé sous forme graphique.

BUS (Numéro_bus, Modèle_bus, Motorisation_bus, Nb_places_bus, Date_mise_serv_bus)

LIGNE (Numéro ligne, Dépard_ligne, Terminus_ligne)

DESSERVIR (#Numéro_bus,# Numéro_ligne)

- Règles de passage du MCD au MLD :

Règle numéro 1 :

a) Une entité du MCD devient une relation, c'est à dire une table.

Dans un SGBD de type relationnel, une table est 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.

b) 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, le fait de définir une clé primaire donne lieu automatiquement à la création d'un index.

c) Les autres propriétés deviennent les attributs de la relation.

CLIENT (numClient, nom, prenom, adresse)

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

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 relation 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.

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

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

III. Structured Query Language SQL

Pour que les différents logiciels et le moteur de base de données puissent se comprendre, ils utilisent un langage appelé SQL.

Ce langage est complet. Il va être utilisé pour :

- Lire les données,
- Ecrire les données,
- Modifier les données,
- Supprimer les données

Il permettra aussi de modifier la structure de la base de données :

- Ajouter des tables,
- Modifier les tables,
- les supprimer
- Ajouter, ou supprimer des utilisateurs,
- Gérer les droits des utilisateurs,
- Gérer les bases de données : en créer de nouvelles, les modifier, etc …

Ce langage est structuré (comme son nom l'indique), c'est à dire que la syntaxe est toujours la même et respecte des normes très précises.

Les avantages du SQL

Comme nous l'avons vu au-dessus, un des principal intérêt du SQL est la **portabilité**. Cela veut dire qu'un logiciel qui utilise une base de données peut fonctionner avec n'importe quelle base de données. Il suffira de lui indiquer avec quelle base de données il doit dialoguer.

Le gros avantage, c'est que si pour une raison X, on doit changer la base, il suffit de modifier la relation entre le logiciel et la base de données. Et hop, le tour est joué !

Je m'adresse ici, à tous les étudiants en informatique:

Du fait de son utilisation sur un très grand nombre de bases de données, une bonne connaissance de SQL est souvent un gros plus sur un CV.

Un autre avantage, est que ce langage étant un langage à part entière, il existe des formations spécifiques et qu'un spécialiste SQL peut très bien intégrer une équipe de développeurs programmant dans différents langages de programmation.

Les inconvénients

Le principal inconvénient est qu'il faut à nouveau apprendre un langage pour utiliser correctement le SQL.

Un autre inconvénient, mais qui n'est pas lié directement au SQL, est que certains éditeurs de logiciels n'hésitent pas à proposer des versions de SQL différentes du standard pour proposer

de nouvelles fonctionnalités ou être le plus près possible de leur base de données. Cela a pour effet de diminuer la portabilité entre les logiciels et les bases de données. (la portabilité est le fait de pouvoir inter changer certaines parties d'un système : le logiciel ou la base de données).

1. Langage de définition des données (LDD) :

Un langage de définition de données est un langage de programmation et un sous-ensemble de SQL pour manipuler les structures de données d'une base de données, et non les données elles-mêmes.

Langage de Définition des Données (LDD) :

- **CREATE** La commande CREATE TABLE permet de créer une table en SQL
- **DROP** La commande DROP TABLE en SQL permet de supprimer définitivement une table d'une base de données
- **ALTER** La commande ALTER TABLE en SQL permet de modifier une table existante.
- **RENAME** La commande RENAME en SQL permet de renommer une table existante.
- **TRUNCATE** La commande TRUNCATE permet de supprimer toutes les données d'une table sans supprimer la table en elle-même.

Langage de Manipulation des Données (LMD) :

- **INSERT** L'insertion de données dans une table s'effectue à l'aide de la commande INSERT INTO. Cette commande permet au choix d'inclure une seule ligne à la base existante ou plusieurs lignes d'un coup
- **UPDATE** La commande UPDATE permet d'effectuer des modifications sur des lignes existantes. Très souvent cette commande est utilisée avec **WHERE** pour spécifier sur quelles lignes doivent porter la ou les modifications
- **DELETE** La commande DELETE en SQL permet de supprimer des lignes dans une table.

Langage d'Interrogation des Données (LID) :

- **SELECT** L'utilisation la plus courante de SQL consiste à lire des données issues de la base de données. Cela s'effectue grâce à la commande SELECT, qui retourne des enregistrements dans un tableau de résultat. Cette commande peut sélectionner une ou plusieurs colonnes
- **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.
- **Projection** Une projection est une instruction permettant de sélectionner un ensemble de colonnes dans une table
- **Sous requête** Les sous-requêtes sont des requêtes qui apparaissent dans une clause **where** ou **having** d'une autre instruction SQL, ou dans la liste de sélection d'une instruction.
- **Alias (AS)** Permet de renommer le nom d'une colonne dans les résultats d'une requête SQL. C'est pratique pour avoir un nom facilement identifiable dans une application qui doit ensuite exploiter les résultats d'une recherche

- **Commande**
  - **UNION** La commande UNION de SQL permet de mettre bout-à-bout les résultats de plusieurs requêtes utilisant elles-mêmes la commande SELECT. C'est donc une commande qui permet de concaténer les résultats de 2 requêtes ou plus
  - **GROUP BY** La commande GROUP BY est utilisée en SQL pour grouper plusieurs résultats et utiliser des fonctions sur le groupement
  - **ORDER BY** La commande ORDER BY permet de trier les lignes dans un résultat d'une requête SQL. Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant ou descendant.
  - **HAVING** La condition HAVING en SQL est presque similaire à WHERE à la seule différence que HAVING permet de filtrer en utilisant des fonctions telles que SUM (), COUNT (), AVG (), MIN () ou MAX ()
- **Fonction**
  - **DISTINCT** Pour éviter des redondances dans les résultats il faut simplement ajouter DISTINCT après le mot SELECT.
  - **COUNT** Lors du développement d'un site ou d'une application il est fort probable de devoir compter certaines informations avec COUNT ()
  - **LIMIT** La clause LIMIT est à utiliser dans une requête SQL pour spécifier le nombre maximum de résultats que l'on souhaite obtenir
  - **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
  - **IN & NOT IN** L'opérateur logique IN ou NOT IN dans SQL s'utilise avec la commande WHERE pour vérifier si une colonne est égale à une des valeurs comprise dans set de valeurs déterminées. C'est une méthode simple pour vérifier si une colonne est égale à une valeur OU une autre valeur OU une autre valeur et ainsi de suite, sans avoir à utiliser de multiple fois l'opérateur OR.
  - **BETWEEN** L'opérateur BETWEEN est utilisé dans une requête SQL pour sélectionner un intervalle de données dans une requête utilisant WHERE
  - **AVG** La fonction d'agrégation AVG () dans le langage SQL permet de calculer une valeur moyenne sur un ensemble d'enregistrement de type numérique et non nul
  - **MIN** La fonction d'agrégation MIN () de SQL permet de retourner la plus petite valeur d'une colonne sélectionnée. Cette fonction s'applique aussi bien à des données numériques qu'à des données alphanumériques.
  - **MAX** La fonction d'agrégation MAX () de SQL permet de retourner la plus petite valeur d'une colonne sélectionnée. Cette fonction s'applique aussi bien à des données numériques qu'à des données alphanumériques.
  - **SUM** Dans le langage SQL, la fonction d'agrégation SUM () permet de calculer la somme totale d'une colonne contenant des valeurs numériques. Cette fonction ne fonction que sur des colonnes de types numériques (INT, FLOAT …)
    et n'additionne pas les valeurs NULL.