SOMMAIRE

Introduction :

Une base de données informatique est un ensemble de données qui ont été stockes sur un support informatique, et organise et structure de manière à pouvoir faciliter exploitation (ajout, mise à jour, recherche de données). Elle se traduit physiquement par un ensemble de fichier disque.

Une base de données permet de filtrer trier classer de large quantité de ‘informations structures concernant un sujet donne et gère par un SGBD. Une base de données seule ne suffit donc pas, il est nécessaire d’avoir également :

- un système permettant de gérer cette base : SGBB
- un langage pour transmettre les instructions a la base de données (par l’intermédiaire du système de gestion) : SQL

Le SGBS :

Un Système de Gestion de Base de Données (SGBD) est un logiciel ou un ensemble de logiciels permettant de manipuler les données d ; une base de données. Manipuler c’est-à-dire sélection et afficher les informations tirées de cette base modifier des données en ajouter ou en supprimer (ce groupe de quatre opération est souvent appelé CRUD) pour Create) Read, Update, Delete). MySQL est un système de gestion de base de données.

MySQL permet de gérer des bases de données donc une grosse quantité d’informations. Il utilise pour cela le langage SQL. Il s’agit d’un des SGBD les plus connues et les plus utilisent.

MySQL peut donc s’utiliser seul, mais la plupart du temps combine a un autre langage de programmation. PHP par exemple.

Cette capacité à gérer les différentes relations porte le nom de Système de Gestion de Base de Données Relationnelle (SGBDR).

Un SGBDR est un SGBD qui implémente la théorie relationnelle. MySQL implémente la théorie du relationnelle ; c’est donc un SGBDR.

Dans un SGBDR, les données sont construites dans ce qu’on appelle des relations, qui sont représentés sous forme de tables. Une relation est compose de deux parties, l’entête et le corps : l’entête est lui-même compose de plusieurs attributs quant au corps il s’agit d’un ensemble de lignes.

En somme : elle réalise les fonctionnalités permettant d’assurer un bon fonctionnement d’une base de données :

- Stockées des données et gestion de l’espace disque
- Gestion du dictionnaire de données
- Recherche et modification des données
- Sécurité, intégrité et confidentialité des données
- Gestion de la concurrence d’accès

MYSQL :

Il s'agit d'un des SGBDR les plus connus et les plus utilises (Wikipédia et adobe utilisent par exemple MYSQL). Et c'est certainement le SGBDR le pus utilise à ce jour pour réaliser des sites web dynamiques.

Le model Serveur-Client

La plupart des SGBD sont bases sur un modèle Client-Serveur. C'est-à-dire que la base de données se trouve sur un serveur qui ne sert qu'à ca et pour interagir avec cette base de données, il faut utiliser un logiciel Client qui va interroger le serveur et transmettre la réponse que le serveur lui aura donné.

Le serveur peut être installe sur une machine différente du client ; c'est souvent le cas lorsque les bases sont souvent importantes.

L'objet le plus représente d'une base de données est la table, chaque table appelé encore relation est caractérisées par une ou plusieurs colonnes (ou attributs). Le langage qui permet de gérer ces objets est appelés « Langage de Description de Données » LDD.

Les données sont stockées dans la table sous formes de ligne ou « tuple ». Le lange qui permet de manipuler ces données est appelé « Langage de Manipulation des Données » LMD

I) Notion Analyse et Conception

Le **système d'information** ou SI, peut être défini comme étant l'ensemble des moyens humains, matériels et immatériels mis en œuvre afin de gérer l'information au sein d'une unité, une entreprise par exemple.

Il ne faut toutefois pas confondre un **système d'information** avec un **système informatique**. En effet, les systèmes d'information ne sont pas toujours totalement informatisés et existaient déjà avant l'arrivée des nouvelles technologies de l'information et des communications dont l'informatique fait partie intégrante.

Le SI possède 4 fonctions essentielles :

- La **saisie** ou **collecte** de l'information
- La **mémorisation** de l'information à l'aide de fichier ou de base de données
- Le **traitement** de l'information afin de mieux l'exploiter (consultation, organisation, mise à jour, calculs pour obtenir de nouvelles données, ...)
- La **diffusion** de l'information

Autrefois, l'information était stockée sur papier à l'aide de formulaires, de dossiers, … et il existait des procédures manuelles pour la traiter. Aujourd'hui, les systèmes informatisés, comme les systèmes de gestion de bases de données relationnelles (SGBDR), sont mis au service du système d'information.

Le système d'information doit décrire (on dit encore représenter) le plus fidèlement possible le fonctionnement du système opérant. Pour ce faire, il doit intégrer une base d'information dans laquelle seront mémorisés la description des objets, des règles et des contraintes du système opérant. Cette base étant sujette à des évolutions, le système d'information doit être doté d'un mécanisme (appelé processeur d'information) destiné à piloter et à contrôler ces changements.

Le processeur d'information produit des changements dans la base d'information à la réception d'un message. Un message contient des informations et exprime une commande décrivant l'action à entreprendre dans la base d'information. Le processeur d'information interprète la commande et effectue le changement en respectant les contraintes et les règles. Si le message exprime une recherche sur le contenu de la base d'information, le processeur interprète la commande et émet un message rendant compte du contenu actuel de la base d'information. Dans tous les cas, l'environnement a besoin de connaître si la commande a été acceptée ou refusée. Le processeur émet, à cet effet, un message vers l'environnement.

Pour aider le concepteur dans ces deux tâches, la méthode Merise propose un ensemble de formalismes et de règles destinées à modéliser de manière indépendante les données et les traitements du système d'information. Ces modèles ne sont qu'une base de réflexion pour le concepteur et un moyen de communication entre les divers acteurs du système d'information dans l'entreprise. Seule la validation de l'ensemble se fera en commun.

La conception d'un système d'information n'est pas évidente car il faut réfléchir à l'ensemble de l'organisation que l'on doit mettre en place. La phase de conception nécessite des méthodes permettant de mettre en place un modèle sur lequel on va s'appuyer. La modélisation consiste à créer une représentation virtuelle d'une réalité de telle façon à faire ressortir les points auxquels on s'intéresse. Ce type de méthode est appelé analyse, il existe plusieurs méthodes d'analyse, la méthode la plus utilisée étant la méthode MERISE.

1. La Méthode MERISE :

Le but de cette méthode est d'arriver à concevoir un système d'information. La méthode MERISE est basée sur la séparation des données et des traitements à effectuer en plusieurs modèles conceptuels et physiques. La séparation des données et des traitements assure une longévité au modèle. En effet, l'agencement des données n'a pas à être souvent remanié, tandis que les traitements le sont plus fréquemment. La méthode MERISE date de 1978-1979.

MERISE est donc une méthode d'analyse et de conception des SI basée sur le principe de la séparation des données et des traitements. Elle possède un certain nombre de **modèles** (ou **schémas**) qui sont répartis sur 3 niveaux :

- Le niveau **conceptuel**,
- Le niveau **logique** ou **organisationnel**,
- Le niveau **physique**.

Même si la méthode MERISE étant, avant tout, une méthode de conception de systèmes d'information, et non de systèmes informatiques, il apparaît aujourd'hui que les systèmes d'information sont largement gérés par des applications informatiques. Les modèles MERISE doivent donc être utilisées pour faciliter le développement de ces applications en s'appuyant sur les technologies logicielles actuelles telles que les bases de données relationnelles et/ou l'architecture client-serveur.

2. La Méthode 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.

Réaliser ces diagrammes revient donc à **modéliser les besoins** du logiciel à développer.

UML est né de la fusion des trois méthodes qui ont influencé la modélisation objet au milieu des années 90 : OMT, Booch et OOSE. Il s'agit d'un compromis qui a été trouvé par une équipe d'experts : Grady Booch, James Rumbaugh et Ivar Jacobson. UML est à présent un standard défini par l'Object Management Group (OMG). De très nombreuses entreprises de renom ont adopté UML et participent encore aujourd'hui à son développement.

UML est surtout utilisé lorsqu'on prévoit de développer des applications avec une démarche objet (développement en Java, en C++, etc.).

Cela dit, je suis d'avis que l'on peut tout à fait s'en servir pour décrire de futures applications, sans pour autant déjà être fixé sur le type de développement.
Le langage UML ne préconise aucune démarche, ce n'est donc pas une méthode. Chacun est libre d'utiliser les types de diagramme qu'il souhaite, dans l'ordre qu'il veut. Il suffit que les diagrammes réalisés soient cohérents entre eux, avant de passer à la réalisation du logiciel.

Modéliser, c'est décrire de manière visuelle et graphique les besoins et, les solutions fonctionnelles et techniques de votre projet logiciel.

Un document de texte décrivant de façon précise ce qui doit être réalisé contiendrait plusieurs dizaines de pages. En général, peu de personnes ont envie de lire ce genre de document. De plus, un long texte de plusieurs pages est source d'interprétations et d'incompréhension. UML nous aide à faire cette **description de façon graphique** et devient alors un excellent moyen pour « **visualiser** » le(s) futur(s) logiciel(s).

3. Etude Comparative :

MERISE (*Méthode d'Etude et de Réalisation Informatique pour les Systèmes d'Entreprise*) est une méthode d'analyse et de réalisation des systèmes d'information qui est élaborée en plusieurs étapes : schéma directeur, étude préalable, étude détaillée et la réalisation.

Alors qu'**UML** (*UnifiedModeling Langage*), est un langage de modélisation des systèmes standard, qui utilise des diagrammes pour représenter chaque aspect d'un système c'est - à - dire : statique, dynamique, ... en s'appuyant sur la notion d'orienté objet qui est un véritable atout pour ce langage.

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 oeuvre.

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...

Après cette étude comparative, on peut utiliser soit l'un soit l'autre selon nos besoins.

II) Concepts Analyse et Conception :

A) MCD :

Il s'agit de l'élaboration du **modèle conceptuel des données** (MCD) qui est une représentation graphique et structurée des informations mémorisées par un SI. Le MCD est basé sur deux notions principales : les **entités** et les **associations**, d'où sa seconde appellation : le **schéma Entité/Association**.

L'élaboration du MCD passe par les étapes suivantes :

- La mise en place de **règles de gestion** (si celles-ci ne vous sont pas données),
- L'élaboration du **dictionnaire des données**,

- La recherche des **dépendances fonctionnelles** entre ces données,
- L'élaboration du MCD (création des **entités** puis des **associations** puis ajout des **cardinalités**).

Avant de vous lancer dans la création de vos tables (ou même de vos entités et associations pour rester dans un vocabulaire conceptuel), il vous faut recueillir les besoins des futurs utilisateurs de votre application. Et à partir de ces besoins, vous devez être en mesure d'établir les règles de gestion des données à conserver.

Ces règles vous sont parfois données mais vous pouvez être amené à les établir vous-même dans deux cas :

Vous êtes à la fois maîtrise d'œuvre (MOE) et maîtrise d'ouvrage (MOA), et vous développez une application pour votre compte et/ou selon vos propres directives.

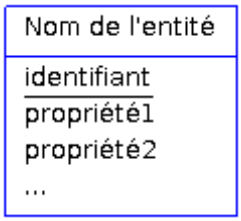
Ce qui arrive le plus souvent : les futurs utilisateurs de votre projet n'ont pas été en mesure de vous fournir ces règles avec suffisamment de précision ; c'est pourquoi vous devrez les interroger afin d'établir vous-même ces règles. N'oubliez jamais qu'en tant que développeur, vous avez un devoir d'assistance à maîtrise d'ouvrage si cela s'avère nécessaire.

- **Les entités :**

Une entité est une représentation d'un ensemble d'objet réel ou abstrait qui ont des caractéristiques communes.

Chaque entité est unique et est décrite par un ensemble de propriétés encore appelées attributs ou caractéristiques. Une des propriétés de l'entité est l'identifiant. Cette propriété doit posséder des occurrences uniques et doit être source des dépendances fonctionnelles avec toutes les autres propriétés de l'entité. Bien souvent, on utilise une donnée de type entier qui s'incrémente pour chaque occurrence, ou encore un code unique spécifique du contexte.

Le formalisme d'une entité est le suivant :



À partir de cette entité, on peut retrouver la règle de gestion suivante en prenant comme exemple la gestion d'une bibliothèque : un auteur est identifié par un numéro unique (id_a) et est caractérisé par un nom, un prénom et une date de naissance.

- **Les attributs :**

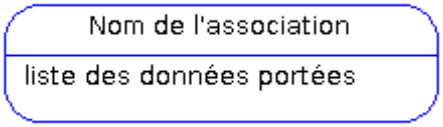
Les attributs correspondent aux caractéristiques des entités c'est-à-dire ce que l'on attribut comme information a ses entités.

Si on se réfère à notre premier exemple les attributs de cette entité sont :

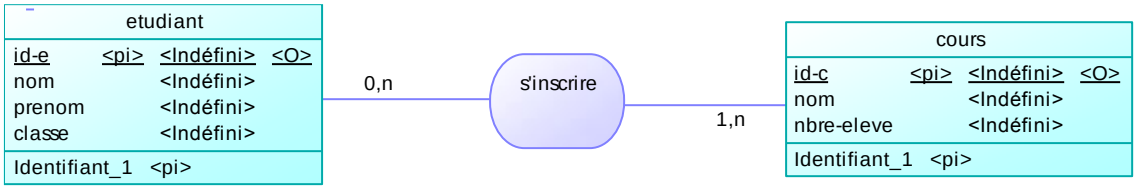
● Les Occurrences

Une association définit un lien sémantique entre une ou plusieurs entités. En effet, la définition de liens entre entités permet de traduire une partie des règles de gestion qui n'ont pas été satisfaites par la simple définition des entités. Une association est généralement nomme a l'aide d'un verbe d'action.

Le formalisme d'une association est le suivant :



Généralement le nom de l'association est un verbe définissant le lien entre les entités qui sont reliées par cette dernière. Par exemple :



Ici l'association «s'inscrire» traduit les deux règles de gestion suivantes :

- Un étudiant ne peut s'inscrire que dans **une classe**,
- Dans une classe, on peut avoir**, un ou plusieurs** étudiants.

Vous remarquerez, que cette association est caractérisée par ces annotations **1,1** et **1,N** qui nous ont permis de définir les règles de gestions précédentes. Ces annotations sont appelées les **cardinalités**.

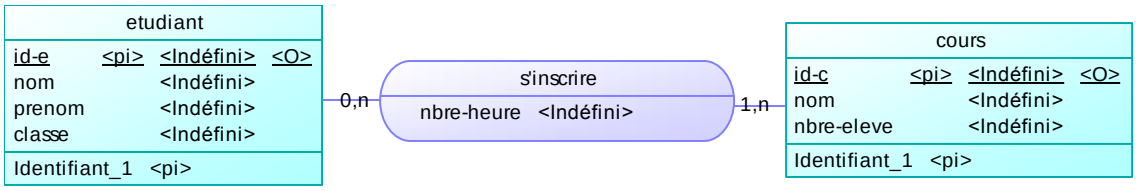
● Les Cardinalités :

Une cardinalité est définie comme ceci :

Minimum, maximum

Les cardinalités les plus répandues sont les suivantes : **0,N** ; **1,N** ; **0,1** ; **1,1**. On peut toutefois tomber sur des règles de gestion imposant des cardinalités avec des valeurs particulières, mais cela reste assez exceptionnel et la présence de ces cardinalités imposera l'implantation de traitements supplémentaires.

L'identifiant d'une association ayant des cardinalités 0,N/1,N de part et d'autre, est obtenu par la concaténation des entités qui participent à l'association. Imaginons l'association suivante :



Ici un étudiant s'inscrit rédige au moins un ou plusieurs classe et pour chaque classe, on connaît le nombre d'heure d'etude (on connaît aussi le nombre total de heure pour chaque cours).

L'association «s'inscrire» peut donc être identifiée par la concaténation des propriétés id-e et id-c. Ainsi, le couple **id-e, id-c** doit être unique pour chaque occurrence de l'association. On peut également définir la dépendance fonctionnelle suivante :

Id-e, idc → nbre_heure

On dit que nbre_heure (nombre d'heure de cours de l'étudiant) est une donnée portée par l'association «s'inscrire». Cette association est donc une association porteuse de données.

Pour une association ayant au moins une cardinalité de type 0,1 ou 1,1 considérons dans un premier temps que cette dernière ne peut être porteuse de données et qu'elle est identifiée par l'identifiant de l'entité porteuse de la cardinalité 0,1 ou 1,1.

Remarque :

- Souvent, pour un même ensemble de règles de gestion, plusieurs solutions sont possibles au niveau conceptuel. Par exemple, rien ne nous obligeait ici à créer une entité **Type**. Une simple donnée portée par l'entité **Livre** aurait pu convenir également.
- Pour que le MCD soit sémantiquement valide, toute entité doit être reliée à au moins une association.
- Les entités et les propriétés peuvent être historiées. Dans ce cas on met un **(H)** à la fin du nom de l'entité ou de la propriété que l'on souhaite historier (cela permet de préciser que l'on archivera toutes les modifications sur une entité ou une propriété donnée). Cela doit également répondre à une règle de gestion.
- Il existe des outils de modélisation payants et d'autres gratuits pour MERISE (powerAMC, OpenModelSphere, AnalyseSI, JMerise, etc).
- On aurait pu, dans ce cas précis, conserver également une date de rentrée des livres, calculée à partir de la date de location et de la durée de celle-ci. C'est un exemple de donnée calculée dont la conservation peut s'avérer pertinente (notamment pour faciliter l'envoi de rappels).

Dans cette partie, nous allons voir comment établir une modélisation des données au niveau logique (ou relationnel) à partir d'un modèle conceptuel, puis comment passer à l'étape de création des tables (cela suppose d'avoir une connaissance préalable des requêtes SQL de création de tables).

B) MLD :

Le modèle logique de données (MLD) est composé uniquement de ce que l'on appelle des **relations**. Ces relations sont à la fois issues des entités du MCD mais aussi d'associations,

dans certains cas. Ces relations nous permettront par la suite de créer nos tables au niveau physique.

Une relation est composée d'attributs. Ces attributs sont des données élémentaires issues des propriétés des différentes entités mais aussi des identifiants et des données portées par certaines associations.

Une relation possède un nom qui correspond en général à celui de l'entité ou de l'association qui lui correspond. Elle possède aussi une **clef primaire** qui permet d'identifier sans ambiguïté chaque occurrence de cette relation. La clef primaire peut être composée d'un ou plusieurs attributs, il s'agit d'une implantation de la notion d'identifiant des entités et associations qui se répercute au niveau relationnel.

Le MLD est souvent représenté de manière textuelle. C'est notamment cette représentation que l'on retrouve dans beaucoup de formations d'études supérieures. Il existe toutefois une représentation graphique équivalente.

Il est important d'accompagner un MLD textuel d'une légende (ce dernier n'ayant pas de formalisme normé). Ceci est d'ailleurs exigé dans certaines formations.

Il existe un autre type de clef appelé **clef étrangère**. La clef étrangère est un attribut d'une relation qui fait référence à la clef primaire d'une autre relation (ces deux clefs devront donc avoir le même type de données).

MLDR: Modèle Logique de Données Relationnelle

- Au niveau relationnel, on devrait plutôt parler de **clef candidate** qui permet d'identifier sans ambiguïté une occurrence de la relation pour les clefs primaires. De même, on devrait désigner une clef étrangère par une **contrainte d'inclusion** vers une clef candidate. Par souci de simplicité, on gardera les termes de clefs primaires et étrangères.
- Par convention, on fait précéder ou suivre la clef étrangère du symbole #. Ceci n'est pas une obligation à partir du moment où les légendes sont suffisamment précises.
- Ici la clef étrangère présente dans la relation «Exemplaire» fait référence à la clef primaire de la relation «Edition».
- Une relation peut posséder aucune, une ou plusieurs clefs étrangères mais possède toujours une et une seule clef primaire.

Enfin, vous pouvez également rencontrer le terme de **cardinalité de la relation** qui signifie ici le nombre d'occurrences d'une relation (ou nombre d'entrées dans la table correspondante) et le terme de **degré de la relation** qui correspond au nombre d'attributs d'une relation.

Comme cela a déjà été dit précédemment, les relations du MLD sont issues des entités du MCD et de certaines associations. Nous allons maintenant aborder ces règles de conversion de façon plus précise.

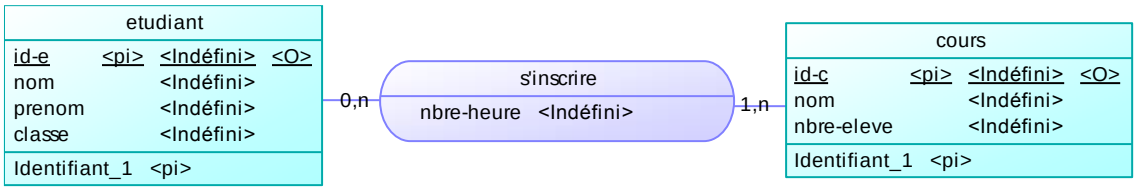
En règle générale, toute entité du MCD devient une relation dont la clef est l'identifiant de cette entité. Chaque propriété de l'entité devient un attribut de la relation correspondante.

Il existe toutefois quelques cas particuliers :

Une association ayant des cardinalités 0,N ou 1,N de part et d'autre devient une relation dont la clef est constituée des identifiants des entités reliées par cette association. Ces identifiants seront donc également des clefs étrangères respectives. On parle de **relations associatives**.

Les cardinalités plus restrictives (comme 2,3 ; 1,7 ; ...) seront perçues comme des cardinalités de type 0/1,N également (il s'agit en effet de sous-ensembles). Cependant, les règles de gestions qui ne seront plus satisfaites par cette modélisation logique devront l'être par des traitements supplémentaires (via le code de l'application qui exploite la base de donnée ou encore par des triggers (déclencheurs) si le SGBDR est suffisamment robuste).

Dans le cas d'associations porteuses de données, les données portées deviennent des attributs de la relation correspondante. Si l'on reprend cet exemple :



L'association «s'inscrire» devrait maintenant être traduite comme ceci en Model Relationnel:

Etudiant (id-e#, nom, prenom, classe)

Cours (id-c#, nom, nbre-eleve)

S'inscrire (*id-e#*, *id-c#*,nb_heure)

Légende:
x :relation
x :clef_primaire
x# : clef étrangère

Plusieurs possibilités s'offrent à nous pour ce cas de figure. La règle de conversion la plus répandue aujourd'hui est d'ajouter une clef étrangère dans la relation qui correspond à l'entité se situant du côté de cette cardinalité 1,1. Cette clef étrangère fera donc référence à la clef de la relation correspondant à la seconde entité reliée par l'association.

Lorsque l'on applique cette règle de conversion, deux restrictions s'imposent :

- L'association ne peut être porteuse de données. Les données portées sont en dépendances fonctionnelles directes avec l'identifiant de l'entité dont la clef correspondante sera référencée par une clef étrangère dans une autre relation.
- L'association doit être binaire (c'est à dire relier uniquement deux entités et pas plus).

Lorsque deux entités sont toutes deux reliées avec une cardinalité 1,1 par une même association, on peut placer la clef étrangère de n'importe quel côté. Par convention, on choisit de la placer du côté de la relation correspondant à l'entité ayant le plus de liaisons avec les autres. Certains considèrent d'ailleurs que deux entités étant reliées par une association ayant une cardinalité 1,1

des deux côtés, doivent obligatoirement fusionner. Cette règle s'appuie encore une fois sur la notion de dépendances fonctionnelles directes mais n'est pas toujours respectée (il est parfois sémantiquement préférable de garder une distinction entre les deux entités).

Une autre solution (moins répandue) consiste à créer une relation associative dont la clef est cette fois composée uniquement de la clef étrangère qui fait référence à l'identifiant de l'entité du côté opposé à la cardinalité 1,1.

Dans ce cas, l'association peut être porteuse de données. Ces dernières deviendront donc des attributs de la relation associative comme dans le cas des cardinalités 0,1/N.

Il va sans dire que la première solution est aujourd'hui préférable à cette dernière en termes d'optimisation et de simplification des requêtes.

De même que pour les cardinalités 1,1, une association ayant une cardinalité 0,1 doit être binaire, et les deux mêmes possibilités s'offrent à nous :

- Créer la clef étrangère dans la relation correspondant à l'entité du côté de la cardinalité 0,1. Rappelons que dans ce cas, l'association ne peut pas être porteuse de données.
- Créer une relation associative qui serait identifié de la même façon que pour une cardinalité 1,1.

Cependant, dans le cadre d'une cardinalité 0,1, nous verrons qu'il n'est pas toujours préférable de privilégier la première méthode comme c'est le cas pour une cardinalité 1,1.

La pertinence de l'une ou l'autre méthode varie en fonction du nombre d'occurrences caractérisées par la cardinalité 0 ou la cardinalité 1. En effet, lorsque les occurrences avec la cardinalité 1 sont plus nombreuses que les occurrences avec la cardinalité 0, la première méthode est préférable. Dans le cas contraire, c'est la seconde méthode qui est la plus adaptée.

Enfin, dans le cas où une association binaire possède à la fois une cardinalité 0,1 et une cardinalité 1,1 (ce qui est rarement le cas), il est préférable que la clef étrangère soit du côté de la relation correspondant à l'entité situé du côté de la cardinalité 1,1.

Avec ces différentes règles de conversion, il nous est déjà possible de convertir notre MCD au complet:

etudiant (id-e,nom,prenom,classe)
cours (id-c,nom,prenom,nbr_elves)
S'inscrire (*id-e#*, *id-c#*)

Légende:
x :relation
x :clefprimaire
x# : clef étrangère

Comme vous pouvez le constater, le schéma de la base est déjà fait. Les règles de passage au SQL sont assez simples :

- chaque relation devient une table
- chaque attribut de la relation devient une colonne de la table correspondante
- chaque clef primaire devient une **PRIMARY KEY**
- chaque clef étrangère devient une **FOREIGN KEY**

IV) SQL :

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

C'est un langage informatique qui permet d'interagir avec des bases de données relationnelles. C'est le langage de base de données le plus répandu, et c'est bien sur celui de MySQL.

C'est donc le langage que nous utilisons pour dire au client MySQL d'effectuer des opérations ou commandes (lire, insérer, modifier et supprimer) sur la base de données stockées sur le serveur MySQL.

Il a été créé dans les années 1970 et c'est devenu un standard en 1986 (pour la norme ANSI et 1987 en ce qui concerne la norme ISO). Il est encore régulièrement amélioré.

Les commandes SQL sont reparties en trois groupes :

✓ Langage de définition des données (LDD) :

- SQL CREATE

La création d'une base de données en SQL est possible en ligne de commande. Même si les systèmes de gestion de base de données (SGBD) sont souvent utilisés pour créer une base, il convient de connaître la commande à utiliser, qui est très simple.

Syntaxe

Pour créer une base de données qui sera appelé ma base il suffit d'utiliser la requête suivante qui est très simple

CREATE DATABASE ma_base

La commande CREATE TABLE permet de créer une table en SQL. Un tableau est une entité qui est contenu dans une base de données pour stocker des données ordonnées dans des colonnes. La création d'une table sert à définir les colonnes et le type de données qui seront contenue dans chacun des colonnes (entier, chaine de caractères, date, valeur binaire…)

La syntaxe générale pour créer une table est la suivante :

CREATE TABLE nom_de_la_table

(colonne 1 type_donnees,colonne 1 type_donne,escolonne 1 type_donnees)

Dans cette requête, 4 colonnes ont été définies. Le mot-clé “type_donnees” sera à remplacer par un mot-clé pour définir le type de données (INT, DATE, TEXT …). Pour chaque colonne, il est également possible de définir des options telles que (liste non-exhaustive):

- **NOT NULL :** empêche d’enregistrer une valeur nulle pour une colonne.
- **DEFAULT :** attribuer une valeur par défaut si aucune donnée n’est indiquée pour cette colonne lors de l’ajout d’une ligne dans la table.
- **PRIMARY KEY :** indiquer si cette colonne est considérée comme clé primaire pour un index.

- SQL ALTER TABLE

En SQL alter table permet de modifier une table existante. Idéal pour ajouter une colonne, supprimer une colonne ou modifier une colonne existante, par exemple pour changer le type.

ALTER TABLE nom_table

Instruction

Le mot-clé “instruction” ici sert à désigner une commande supplémentaire, qui sera détaillée ci-dessous selon l’action que l’on souhaite effectuer : ajouter, supprimer ou modifier une colonne.

Ajouter une colonne

L’ajout d’une colonne dans une table est relativement simple et peut s’effectuer à l’aide d’une requête ressemblant à ceci:

ALTER TABLE nom_table

ADD nom_colonne type_donnees

Exemple
Pour ajouter une colonne qui correspond à une rue sur une table utilisateur, il est possible d’utiliser la requête suivante:

ALTER TABLE Users

ADD address_rue varchar (255)

Supprimer une colonne

Une syntaxe permet également de supprimer une colonne pour une table. Il y’a 2 manières totalement équivalentes pour supprimer une colonne

ALTER TABLE nom table

DROP nom colonne

- SQL TRUNCATE

En SQL, la commande TRUNCATE permet de supprimer toutes les données d’une table sans supprimer la table en elle-même. En d’autres mots, cela permet de purger la table.

Cette instruction diffère de la commande DROP qui a pour but de supprimer les données ainsi que la table qui les contient.

A noter : l'instruction TRUNCATE est semblable à l'instruction DELETE sans utilisation de WHERE. Parmi les petites différences TRUNCATE est toutefois plus rapide et utilise moins de ressource. Ces gains en performance se justifie notamment parce que la requête n'indiquera pas le nombre d'enregistrement supprimés et qu'il n'y aura pas d'enregistrement des modifications dans le journal.

Cette instruction s'utilise dans une requête SQL semblable à celle-ci :

TRUNCATE TABLE 'table'

Dans cet exemple, les données de la table "table" seront perdues une fois cette requête exécutée.

✓ Langage de manipulation de données (LMD) :

- SQL 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.

La syntaxe basique d'une requête utilisant UPDATE est la suivante :

Update table

Set nom_colonne_1 = 'nouvelle valeur'

WHERE condition

Cette syntaxe permet d'attribuer une nouvelle valeur à la colonne nom_colonne_1 pour les lignes qui respectent la condition stipulé avec WHERE. Il est aussi possible d'attribuer la même valeur à la colonne nom_colonne_1 pour toutes les lignes d'une table si la condition WHERE n'était pas utilisé.

A noter, pour spécifier en une seule fois plusieurs modification, il faut séparer les attributions de valeur par des virgules. Ainsi la syntaxe deviendrait la suivante :

Update table

Set colonne_1 = 'valeur1', colonne_ 2= 'valeur2', colonne_3 = 'valeur3'

WHERE condition

- SQL DELETE

La commande DELETE en SQL permet de supprimer des lignes dans une table. En utilisant cette commande associe à ZHERE il est possible de sélectionner les lignes concernées au seront supprimés.

La syntaxe pour supprimer des lignes est la suivante :

DELETE FROM 'table'

WHERE condition

- **SQL INSERT INTO**

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.

Pour insérer des données dans une base, il y 'a 2 syntaxes principales :

Insérer une ligne en indiquant les informations pour chaque colonne existante (en respectant l'ordre)

Insérer une ligne en spécifiant les colonnes que vous souhaitez compléter. Il est possible d'insérer une ligne renseignant seulement une partie des colonnes.

Insérer une ligne en spécifiant toutes les colonnes

La syntaxe pour remplir une ligne avec cette méthode est la suivante :

INSERT INTO table VALUES ('valeur1', 'valeur2',…)

Cette syntaxe possède les avantages et inconvénients suivants :

- Obliger de remplir toutes les données, tout en respectant l'ordre des colonnes
- Il n'y a pas le nom de colonne, donc les fautes de frappe sont limitées. Par ailleurs, les colonnes peuvent être renommées sans avoir à changer la requête
- L'ordre des colonnes doit rester identique sinon certaines valeurs prennent le risque d'être complétée dans la mauvaise colonne

Insérer une ligne en spécifiant seulement les colonnes souhaitées

Cette deuxième solution est très similaire, excepté qu'il faut indiquer le nom des colonnes avant "VALUES". La syntaxe est la suivante :

INSERT INTO table (nom_colonne1, nom_colonne2,…

VALUES ('valeur1', 'valeur2',…)).

✓ Langage d'interrogation de données (LID) : SELECT

- **SQL 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 d'une table.

L'utilisation basique de cette commande s'effectue de la manière suivante:

SELECT nom_du_champs FROM nom_du_tableau

Cette requête SQL va **sélectionner** (SELECT) le champ "nom_du_champ" **provenant** (FROM) du tableau appelé "nom_du_tableau".

✓ Les commandes :

- La commande 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.

Syntaxe

Une requête où l'on souhaite filtrer l'ordre des résultats utilise la commande ORDER BY de la sorte :

```
SELECT colonne1, colonne2

FROM table

ORDER BY colonne2
```

Par défaut les résultats sont classés par ordre ascendant, toutefois il est possible d'inverser l'ordre en utilisant le suffixe DESC après le nom de la colonne. Par ailleurs, il est possible de trier sur plusieurs colonnes en les séparant par une virgule. Une requête plus élaborée ressemblerait à cela :

```
SELECT colonne1, colonne2

FROM table

ORDER BY colonne1 DESC, colonne2 ASC
```

A noter : il n'est pas obligé d'utiliser le suffixe "ASC" sachant que les résultats sont toujours classés par ordre ascendant par défaut. Toutefois, c'est plus pratique pour mieux s'y retrouver, surtout si on a oublié l'ordre par défaut

- **La commande UNION**

La commande UNION de SQL permet de mettre bout-à-bout les résultats de plusieurs requêtes utilisant elles-même la commande SELECT. C'est donc une commande qui permet de concaténer les résultats de 2 requêtes ou plus. Pour l'utiliser il est nécessaire que chacune des requêtes à concaténer retournes le même nombre de colonnes, avec les mêmes types de données et dans le même ordre.

A savoir : par défaut, les enregistrements exactement identiques ne seront pas répétés dans les résultats. Pour effectuer une union dans laquelle même les lignes dupliquées sont affichées il faut plutôt utiliser la commande UNION ALL.

Syntaxe

La syntaxe pour unir les résultats de 2 tableaux sans afficher les doublons est la suivante:

```
SELECT * FROM table1

Union
```

```
SELECT * FROM table2
```

- **La commande 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().

Syntaxe

L'utilisation de HAVING s'utilise de la manière suivante :

```
SELECT colonne1, SUM (colonne2)

FROM nom_table

GROUP BY colonne1

HAVING function (colonne2) operateur valeur
```

Cela permet donc de SÉLECTIONNER les colonnes DE la table "nom_table" en GROUPANT les lignes qui ont des valeurs identiques sur la colonne "colonne1" et que la condition de HAVING soit respectée.

Important : HAVING est très souvent utilisé en même temps que GROUP BY bien que ce ne soit pas obligatoire.

- **La commande GROUP BY**

La commande GROUP BY est utilisée en SQL pour grouper plusieurs résultats et utiliser une fonction de totaux sur un groupe de résultat. Sur une table qui contient toutes les ventes d'un magasin, il est par exemple possible de lister regrouper les ventes par clients identiques et d'obtenir le coût total des achats pour chaque client.

Syntaxe d'utilisation de GROUP BY

De façon générale, la commande GROUP BY s'utilise de la façon suivante

```
SELECT colonne1, fonction (colonne2)

FROM table

GROUP BY
```

A noter : cette commande doit toujours s'utiliser après la commande WHERE et avant la Commande HAVING.

- ✓ Les fonctions d'agrégation :

- **La fonction COUNT ()**

En SQL, la fonction d'agrégation COUNT () permet de compter le nombre d'enregistrement dans une table. Connaître le nombre de lignes dans une table est très pratique dans de nombreux cas, par exemple pour savoir combien d'utilisateurs sont présents dans une table ou pour connaître le nombre de commentaires sur un article.

Pour connaître le nombre de lignes totales dans une table, il suffit d'effectuer la requête SQL suivante :

```
SELECT COUNT (*) FROM table
```

Il est aussi possible de connaitre le nombre d'enregistrement sur une colonne en particulier. Les enregistrements qui possèdent la valeur nulle ne seront pas comptabilisé. La syntaxe pour compter les enregistrements sur la colonne "nom_colonne" est la suivante :

```
SELECT COUNT (nom_colonne) FROM table
```

Enfin, il est également possible de compter le nombre d'enregistrement distinct pour une colonne. La fonction ne comptabilisera pas les doublons pour une colonne choisie. La syntaxe pour compter le nombre de valeur distincte pour la colonne "nom_colonne" est la suiante :

```
SELECT COUNT (DISTINCT nom_colonne) FROM table
```

- **La fonction 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.

Syntaxe

La syntaxe pour utiliser cette fonction de statistique est simple :

```
SELECT AVG (nom_colonne) FROM nom_table
```

Cette requête permet de calculer la note moyenne de la colonne "nom_colonne" sur tous les enregistrements de la table "nom_table". Il est possible de filtrer les enregistrements concernés à l'aide de la commande WHERE. Il est aussi possible d'utiliser la commande GROUP BY pour regrouper les données appartenant à la même entité.
A savoir : la syntaxe est conforme avec la norme SQL et fonctionne correctement avec tous les Systèmes de Gestion de Base de Données (SGBD), incluant : MySQL, PostgreSQL, Oracle et SQL Server.

- **La fonction MAX ()**

Dans le langage SQL, la fonction d'agrégation MAX() permet de retourner la valeur maximale d'une colonne dans un set d'enregistrement. La fonction peut s'appliquée à des données numériques ou alphanumériques. Il est par exemple possible de rechercher le produit le plus cher dans une table d'une boutique en ligne.

La syntaxe de la requête SQL pour retourner la valeur maximum de la colonne "nom_colonne" est la suivante:

```
SELECT MAX (nom_colonne) FROM table
```

Lorsque cette fonctionnalité est utilisée en association avec la commande GROUP BY, la requête peut ressembler à l'exemple ci-dessous:

```
SELECT colonne1, MAX(colonne2)

FROM table

GROUP BY colonne1
```

- **La fonction 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.

Pour obtenir la plus petite valeur de la colonne "nom_colonne" il est possible d'utiliser la requête SQL suivante:

```
SELECT MIN (nom_colonne) FROM table
```

Étant données qu'il s'agit d'une fonction d'agrégation, il est possible de l'utiliser en complément de la commande GROUP BY. Cela permet de grouper des colonnes et de connaître la plus petite valeur pour chaque groupe. La syntaxe est alors la suivante:

```
SELECT colonne1, MIN(colonne2)

FROM table

GROUP BY colonne1
```

Cet exemple permet de grouper tous les enregistrements de "colonne1" de la table et de connaître la plus petite valeur de "colonne2" pour chacun de ces regroupement.

- **La fonction 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.

La syntaxe pour utiliser cette fonction SQL peut être similaire à celle-ci:

```
SELECT SUM (nom_colonne) FROM table
```

Cette requête SQL permet de calculer la somme des valeurs contenu dans la colonne "nom_colonne".

A savoir : Il est possible de filtrer les enregistrements avec la commande WHERE pour ne calculer la somme que des éléments souhaités.

- **La fonction 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 multiples.

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 enregistrements 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".
- 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".

- **La fonction BETWEEN**

L'opérateur BETWEEN est utilisé dans une requête SQL pour sélectionner un intervalle de données dans une requête utilisant WHERE. L'intervalle peut être constitué de chaînes de caractères, de nombres ou de dates. L'exemple le plus concret consiste par exemple à récupérer uniquement les enregistrements entre 2 dates définies.

Syntaxe

L'utilisation de la commande BETWEEN s'effectue de la manière suivante :

SELECT * FROM table

WHERE nom_colonne BETWEEN 'valeur1' AND 'valeur2'

La requête suivante retournera toutes les lignes dont la valeur de la colonne "nom_colonne" sera comprise entre **valeur1** et **valeur2**.

- **La fonction IN**

L'opérateur logique 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.

Syntaxe

Pour chercher toutes les lignes où la colonne "nom_colonne" est égale à 'valeur 1' OU 'valeur 2' ou 'valeur 3', il est possible d'utiliser la syntaxe suivante:

SELECT nom_colonne FROM table

WHERE nom_colonne IN (valeur1, valeur2, valeur3, …)

A savoir : entre les parenthèses il n'y a pas de limite du nombre d'arguments. Il est possible d'ajouter encore d'autres valeurs.
Cette syntaxe peut être associée à l'opérateur NOT pour recherche toutes les lignes qui ne sont pas égales à l'une des valeurs stipulées.

Simplicité de l'opérateur IN

La syntaxe utilisée avec l'opérateur est plus simple que d'utiliser une succession d'opérateur OR. Pour le montrer concrètement avec un exemple, voici 2 requêtes qui retournerons les mêmes résultats, l'une utilise l'opérateur IN, tandis que l'autre utilise plusieurs OR.

Requête avec plusieurs OR :

SELECT prenom FROM users
WHERE prenom = 'asta' OR premon = 'ibou' OR prenom = 'maria'

Requête équivalent avec l'opérateur IN :

SELECT prenom FROM users
WHERE prenom IN ('ibou', 'maria', 'asta')

- **La commande AS (alias)**

Dans le langage SQL il est possible d'utiliser des **alias** pour renommer temporairement une colonne ou une table dans une requête. Cette astuce est particulièrement utile pour faciliter la lecture des requêtes.

Intérêts et utilités

Alias sur une colonne

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.

Alias sur une table

Permet d'attribuer un autre nom à une table dans une requête SQL. Cela peut aider à avoir des noms plus court, plus simple et plus facilement compréhensible. Ceci est particulièrement vrai lorsqu'il y a des jointures.

Syntaxe

Alias sur une colonne

La syntaxe pour renommer une colonne de **colonne1** à **c1** est la suivante:

```
SELECT colonne1 AS cl, colonne2
FROM table
```

Cette syntaxe peut également s'afficher de la façon suivante:

```
SELECT colonne1  cl, colonne2
FROM table
```

A noter : Pour choisir il est préférable d'utiliser la commande "**AS**" pour que ce soit plus explicite (plus simple à lire qu'un simple espace), d'autant plus que c'est recommandé dans le standard ISO pour concevoir une requête SQL.

Alias sur une table

La syntaxe pour renommer une table dans une requête est la suivante:

```
SELECT * FROM 'nom_table' AS t1
```

Cette requête peut également s'écrire de la façon suivante:

```
SELECT * FROM 'table'
```

Source :

https://sql.sh/cours

https://openclassrooms.com/fr/courses/2035826-debutez-lanalyse-logicielle-avec-uml/2035851-uml-c-est-quoi,

https://www.youtube.com/watch?v=jVDe7SAp43k&list=PLcOmJ-JvAV1dhlZrmJ3XiC_oLbeJCOg81&index=13