GUIDE

INTRODUCTION AUX BASES DE DONNÉES

Définition

Il est difficile de donner une définition exacte de la notion de base de données. Une définition très générale pourrait être : un ensemble organisé d'informations avec un objectif commun.
Peu importe le support utilisé pour rassembler et stocker les données (papier, fichiers, etc.), dès lors que des données sont rassemblées et stockées d'une manière organisée dans un but spécifique, on parle de base de données.
Plus précisément, on appelle base de données un ensemble structuré et organisé permettant le stockage de grandes quantités d'informations afin d'en faciliter l'exploitation (ajout, mise à jour, recherche de données). Et dans le domaine qui nous intéresse, ce serait un ensemble structuré de données enregistrées sur des supports accessibles par l'ordinateur, représentant des informations du monde réel et pouvant être interrogées et mises à jour par une communauté d'utilisateurs.

Modèle de base de données

Le résultat de la conception d'une base de données informatisée est une description des données. Par description on entend définir les propriétés d'ensembles d'objets modélisés dans la base de données et non pas d'objets particuliers. Les objets particuliers sont créés par des programmes d'applications ou des langages de manipulation lors des insertions et des mises à jour des données.
Cette description des données est réalisée en utilisant un modèle de données. Ce dernier est un outil formel utilisé pour comprendre l'organisation logique des données. Parmi ces modèles, on distingue:

- **Le modèle hiérarchique:** une base de données hiérarchique est une forme de système de gestion de base de données qui lie des enregistrements dans une structure arborescente de façon à ce que chaque enregistrement n'ait qu'un seul possesseur (par exemple, une paire de chaussures n'appartient qu'à une seule personne).
- **Le modèle réseau:** Le modèle réseau est en mesure de lever de nombreuses difficultés du modèle hiérarchique grâce à la possibilité d'établir des liaisons de type n-n, les liens entre objets pouvant exister sans restriction. Pour retrouver une donnée dans une telle modélisation, il faut connaître le chemin d'accès (les liens) ce qui rend les programmes dépendants de la structure de données

- **Le modèle relationnel**: Les bases de données relationnelles ont été inventées en 1970 par E.F. Codd de IBM. Il s'agit de documents tabulaires dans laquelle les données sont définies afin d'être accessibles et de pouvoir être réorganisées de différentes manières. Les bases de données relationnelles sont constituées d'un ensemble de tableaux. Au sein de ces tableaux, les données sont classées par catégorie. Chaque tableau comporte au moins une colonne correspondant à une catégorie. Chaque colonne comporte un certain nombre de données correspondant à cette catégorie. L'API standard pour les bases de données relationnelles est **le Structured Query Language (SQL)**. Les bases de données relationnelles sont facilement extensibles, et de nouvelles catégories de données peuvent être ajoutées après la création de la « database » originale sans avoir besoin de modifier toutes les applications existantes.

SGBD

La gestion et l'accès à une base de données sont assurés par un ensemble de programmes qui constituent le *Système de gestion de base de données* (SGBD). Un SGBD est caractérisé par le modèle de description des données qu'il supporte (hiérarchique, réseau, relationnel, objet, etc.). Les données sont décrites sous la forme de ce modèle, grâce à un Langage de Description des Données (LDD). Cette description est appelée *schéma*.

Une fois la base de données spécifiée, on peut y insérer des données, les récupérer, les modifier et les détruire. C'est ce qu'on appelle manipuler les données. Les données peuvent être manipulées non seulement par un Langage spécifique de Manipulation des Données (LMD), mais aussi par des langages de programmation classiques.

Un SGBD doit permettre l'ajout, la modification et la recherche de données. Un système de gestion de bases de données héberge généralement plusieurs bases de données, qui sont destinées à des logiciels ou des thématiques différents.

Quel que soit le modèle, un des problèmes fondamentaux à prendre en compte est la cohérence des données. Par exemple, dans un environnement où plusieurs utilisateurs peuvent accéder concurremment à une colonne d'une table par exemple pour la lire ou pour l'écrire, il faut s'accorder sur la politique d'écriture. Cette politique peut être : les lectures concurrentes sont autorisées, mais dès qu'il y a une écriture dans une colonne, l'ensemble de la colonne est envoyé aux autres utilisateurs l'ayant lue pour qu'elle soit rafraîchie.

Des objectifs principaux ont été fixés aux SGBD dès l'origine de ceux-ci, et ce, afin de résoudre les problèmes causés par la démarche classique. Ils sont entre autres: l’indépendance physique, l’indépendance logique, l’accès aux données, la non-redondance des données etc.

Enfin, parmi les SGBD les plus utilisés, on distingue: PostegreSQL, MySQL, Oracle, etc.

CONCEPTION DE BASE DE DONNÉES

Modélisation

Il est difficile de modéliser un domaine sous une forme directement utilisable par un SGBD. Une ou plusieurs modélisations intermédiaires sont donc utiles, le modèle entités-associations constitue l'une des premières et des plus courantes. Ce modèle permet une description naturelle du monde réel à partir des concepts d'entité et d'association. Basé sur la théorie des ensembles et des relations, ce modèle se veut universel et répond à l'objectif d'indépendance données-programmes. Ce modèle, utilisé pour la phase de conception, s'inscrit notamment dans le cadre d'une méthode plus générale et très répandue : *Merise*

Merise

MERISE (Méthode d'Étude et de Réalisation Informatique pour les Systèmes d'Entreprise) est certainement le langage de spécification le plus répandu dans la communauté de l'informatique des systèmes d'information, et plus particulièrement dans le domaine des bases de données. Une représentation Merise permet de valider des choix par rapport aux objectifs, de quantifier les solutions retenues, de mettre en œuvre des techniques d'optimisation et enfin de guider jusqu'à l'implémentation. Reconnu comme standard, Merise devient un outil de communication. En effet, Merise réussit le compromis difficile entre le souci d'une modélisation précise et formelle, et la capacité d'offrir un outil et un moyen de communication accessible aux non-informaticiens. Un des concepts clés de la méthode Merise est la séparation des données et des traitements. Cette méthode est donc parfaitement adaptée à la modélisation des problèmes abordés d'un point de vue fonctionnel. Les données représentent la statique du système d'information et les traitements sa dynamique. L'expression conceptuelle des données conduit à une modélisation des données en entités et en associations. Dans ce cours, nous écartons volontairement la modélisation des traitements puisque nous ne nous intéressons à la méthode Merise que dans la perspective de la modélisation de bases de données.

La démarche de Merise

Merise propose une démarche, dite par niveaux, dans laquelle il s'agit de hiérarchiser les préoccupations de modélisation qui sont de trois ordres : la conception, l'organisation et la technique. En effet, pour aborder la modélisation d'un système, il convient de l'analyser en premier lieu de façon globale et de se concentrer sur sa fonction : c'est-à-dire de s'interroger sur ce qu'il fait avant de définir comment il le fait. Ces niveaux de modélisation sont organisés dans une double approche données/traitements. Les trois niveaux de représentation des données, puisque ce sont eux qui nous intéressent, sont détaillés ci-dessous:

- **Niveau conceptuel :** le *modèle conceptuel des données (MCD)* décrit les entités du monde réel, en terme d'objets, de propriétés et de relations, indépendamment de toute technique d'organisation et d'implantation des données. Ce modèle se concrétise par un *schéma entités-associations* représentant la structure du système d'information, du point de vue des données.
- **Niveau logique :** le *modèle logique des données (MLD)* précise le modèle conceptuel par des choix organisationnels. Il s'agit d'une transcription (également appelée dérivation) du MCD dans un formalisme adapté à une implémentation ultérieure, au niveau physique, sous forme de base de données relationnelle ou réseau, ou autres. Les choix techniques d'implémentation (choix d'un SGBD) ne seront effectués qu'au niveau suivant.
- **Niveau physique :** le *modèle physique des données (MPD)* permet d'établir la manière concrète dont le système sera mis en place (SGBD retenu).

Éléments constitutifs du modèle entités-associations

La représentation du modèle entités-associations s'appuie sur trois concepts de base :

- l'objet ou entité ;
- l'association ;
- la propriété.

L'objet est une entité ayant une existence propre. L'association est un lien ou relation entre objets sans existence propre. La propriété est la plus petite donnée d'information décrivant un objet ou une association.

Entité

Une entité est un objet, une chose concrète ou abstraite qui peut être reconnue distinctement et qui est caractérisée par son unicité. Exemples: Fatou Ndiaye, le livre que je tiens entre les mains, ma voiture, etc.

Les entités ne sont généralement pas représentées graphiquement.

Un type entité maintenant désigne un ensemble d'entités qui possèdent une sémantique et des propriétés communes. Exemples: les personnes, les livres, les voitures etc. En effet, dans le cas d'une personne par exemple, les informations associées (*i.e. les propriétés*), comme le nom et le prénom, ne changent pas de nature.

Une entité est souvent nommée occurrence ou instance de son *type entité*.

les types *entité Personne*, caractérisée par un nom et un prénom, et *Voiture*, caractérisée par un nom et une puissance fiscale, ne peuvent pas être regroupées, car ils ne partagent pas leurs propriétés (le prénom est une chaîne de caractères et la puissance fiscale un nombre). les types *entité Personne*, caractérisés par un nom et un prénom, et *Livre*, caractérisé un titre et un auteur, possèdent chacun, deux attributs du type *chaîne de caractères*. Pourtant, ces deux *type entité* ne peuvent pas être regroupés, car ils ne partagent pas une même sémantique : le nom d'une personne n'a rien à voir avec le titre d'un livre, le prénom d'une personne n'a rien à voir avec un auteur.

Attribut, propriété ou valeur

Un attribut (ou une propriété) est une caractéristique associée à un type entité ou à un type association. Exemples: le nom d'une personne, le titre d'une livre, la puissance d'une voiture.

Au niveau du type entité ou du type association, chaque attribut possède un domaine qui définit l'ensemble des valeurs possibles qui peuvent être choisies pour lui (entier, chaîne de caractères, booléen…). Au niveau de l'entité, chaque attribut possède une valeur compatible avec son domaine.

- **Règle 1:** Un attribut ne peut en aucun cas être partagé par plusieurs type entité ou type association.
- **Règle 2:** Un attribut est une donnée élémentaire, ce qui exclut des données calculées ou dérivées.
- **Règle 3:** Un type entité et ses attributs doivent être cohérents entre eux (*i.e.* ne traiter que d'un seul sujet).

Par exemple, si le modèle doit comporter des informations relatives à des articles et à leur fournisseur, ces informations ne doivent pas coexister au sein d'un même *type entité*. Il est préférable de mettre les informations relatives aux articles dans un *type entité Article* et les informations relatives aux fournisseurs dans un *type entité Fournisseur*. Ces deux *type entité* seront probablement ensuite reliés par un *type association*.

Identifiant ou clé primaire

Un identifiant (ou clé) d'un type entité ou d'un type association est constitué par un ou plusieurs de ses attributs qui doivent avoir une valeur unique pour chaque entité ou association de ce type. Il est donc impossible que les attributs constituant l'identifiant d'un *type entité* (respectivement *type association*) prennent la même valeur pour deux entités (respectivement deux associations) distinctes. Exemples d'identifiant : le numéro de sécurité sociale pour une personne, le numéro d'immatriculation pour une voiture, le code ISBN d'un livre pour un livre (mais pas pour un exemplaire).

Règle: Chaque type entité possède au moins un identifiant, éventuellement formé de plusieurs attributs.

Ainsi, chaque *type entité* possède au moins un attribut qui, s'il est seul, est donc forcément l'identifiant.

Dans la représentation graphique, les attributs qui constituent l'identifiant sont soulignés et placés en tête.

Association ou relation

Une association (ou une relation) est un lien entre plusieurs entités. Exemple : l'emprunt par l'étudiant Arame du 3e exemplaire du livre « Maîtrisez SQL ». Les associations ne sont généralement pas représentées graphiquement.

Un type association (ou un type relation) désigne un ensemble de relations qui possèdent les mêmes caractéristiques. Le type association décrit un lien entre plusieurs type entité. Les associations de ce type association lient des entités de ces type entité. Comme les types *entité*, les types *association* sont définis à l'aide d'attributs qui prennent leur valeur dans les associations.

Règle: Un attribut peut être placé dans un type association uniquement lorsqu'il dépend de toutes les entités liées par le type association.

Un *type association* peut ne pas posséder d'attribut explicite et cela est relativement fréquent, mais on verra qu'il possède au moins des attributs implicites. Exemple: l'emprunt d'un livre à la bibliothèque.

Une association est souvent nommée occurrence ou instance de son *type association*.

- **Définition 1 -participant-** les types entité intervenant dans un type association sont appelés les participants de ce type association.
- **Définition 2 -collection-** L'ensemble des participants d'un type association est appelé la collection de ce type association.
- **Définition 3 -dimension ou arité d'un type association-** La dimension, ou l'arité d'un type association est le nombre de type entité contenu dans la collection.

Comme un *type entité*, un *type association* possède forcément un identifiant, qu'il soit explicite ou non.

Règle: La concaténation des identifiants des types entité liés à un type association constitue un identifiant de ce type association et cet identifiant n'est pas mentionné sur le modèle (il est implicite). Cette règle implique que deux instances d'un même *type association* ne peuvent lier un même ensemble d'entités.

Souvent, un sous-ensemble de la concaténation des identifiants des types *entité* liés suffit à identifier le *type association*.

On admet également un identifiant plus naturel et explicite, à condition qu'il ne soit qu'un moyen d'exprimer plus simplement cette concaténation.

Cardinalité

La cardinalité d'une patte reliant un type association et un type entité précise le nombre de fois minimal et maximal d'interventions d'une entité du type entité dans une association du type association. La cardinalité minimale doit être inférieure ou égale à la cardinalité maximale. Exemple: une personne peut être l'auteur de 0 à *n* livre, mais un livre ne peut être écrit que par une personne.

Règle: L'expression de la cardinalité est obligatoire pour chaque patte d'un type association.

Une cardinalité minimale est toujours 0 ou 1 et une cardinalité maximale est toujours 1 ou n.

Ainsi, si une cardinalité maximale est connue et vaut 2, 3 ou plus, alors nous considérons qu'elle est indéterminée et vaut *n*. En effet, si nous connaissons *n* au moment de la conception, il se peut que cette valeur évolue au cours du temps. Il vaut donc mieux considérer *n* comme inconnue dès le départ. De la même manière, on ne modélise pas des cardinalités minimales qui

valent plus de 1, car ces valeurs sont également susceptibles d'évoluer. Enfin, une cardinalité maximale de 0 n'a pas de sens, car elle rendrait le *type association* inutile.
Les seules cardinalités admises sont donc :

- **0,1 :**

une occurrence du *type entité* peut exister tout en n'étant impliquée dans aucune association et peut être impliquée dans au maximum une association.

- **0,n :**

c'est la cardinalité la plus ouverte ; une occurrence du *type entité* peut exister tout en n'étant impliquée dans aucune association et peut être impliquée, sans limitation, dans plusieurs associations.

- **1,1 :**

une occurrence du *type entité* ne peut exister que si elle est impliquée dans exactement (au moins et au plus) une association.

- **1,n :**

une occurrence du *type entité* ne peut exister que si elle est impliquée dans au moins une association.

MCD & MLD

Présentation

Le MCD est la représentation la plus abstraite que l'on réalise de la structure des données d'un système d'information. Il est constitué d'entités qui représentent des ensembles de données de même nature et d'associations entre entités.

Le MLD est une représentation qui prend en compte le choix technologique de la réalisation de la future base de données. Ainsi, il est aussi appelé modèle relationnel lorsqu'on travaille avec une base de données relationnelle.

Et quelquefois, les abréviations suivantes sont employées :

- MLDR : Modèle logique de données relationnelles
- MRD : Modèle relationnel de données
- MLRD : Modèle relationnel logique de données

Le MCD (Modèle Conceptuel de Données) ne peut pas être implanté dans une base de données sans modification. Il est obligatoire de transformer ce modèle. On dit qu'on effectue un passage du modèle conceptuel de données vers le modèle logique de données.

Règles de passage du MCD au MLD

Règle 1:

- **Une entité du MCD devient une relation, c'est à dire une table**. Dans un SGBD (Système de Gestion de base de données) de type relationnel, une table est une structure tabulaire dont chaque ligne correspond aux données d'un objet enregistré (d'où le terme enregistrement) et où chaque colonne correspond à une propriété de cet objet. Une table contiendra donc un ensemble d'enregistrements. Une ligne correspond à un enregistrement. Une colonne correspond à un champ. La valeur prise par un champ pour un enregistrement donné est située à l'intersection ligne-colonne correspondant à enregistrement-champ. Il n'y a pas de limite théorique au nombre d'enregistrements que peut contenir une table. Par contre, la limite est liée à l'espace de stockage.
- **Son identifiant devient la clé primaire de la relation**. La clé primaire permet d'identifier de façon unique un enregistrement dans la table. Les valeurs de la clé primaire sont donc uniques. Les valeurs de la clé primaire sont obligatoirement non nulles. Dans la plupart des SGBDR (Système de Gestion de Base de Données Relationnelle), le fait de définir une clé primaire donne lieu automatiquement à la création d'un index. Un index est un fichier interne au SGBD. L'utilisateur standard n'a

pas besoin d'y accéder. L'index a pour but d'accélérer les traitements de recherche, de tri, de filtre et notamment sur les tables avec de nombreux enregistrements. La contrepartie est que l'index nécessite de l'espace mémoire et surtout, les temps d'insertion, de suppression d'enregistrements sont plus importants car il faut mettre à jour à la fois la table et l'index.

- **Les autres propriétés deviennent les attributs de la relation.**
- **Règle 2:**
- Une association de type 1:N (c'est à dire qui a les cardinalités maximales positionnées à « 1 » d'une côté de l'association et à « n » de l'autre côté) se traduit par la création d'une clé étrangère dans la relation correspondante à l'entité côté « 1 ». Cette clé étrangère référence la clé primaire de la relation correspondant à l'autre entité.
- **Règle 3:**
- Une association de type N :N (c'est à dire qui a les cardinalités maximales positionnées à « N » des 2 côtés de l'association) se traduit par la création d'une table dont la clé primaire est composée des clés étrangères référençant les relations correspondant aux entités liées par l'association. Les éventuelles propriétés de l'association deviennent des attributs de la relation.

UML : Unified Modeling Language

Présentation

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. Ces diagrammes sont au nombre de onze on distingue parmi eux, le diagramme de classe et le diagramme de cas d'utilisation.

UML se base sur l'approche objet qui est une démarche qui s'organise autour de 4 principes fondamentaux. C'est une démarche :

- itérative et incrémentale ;
- guidée par les besoins du client et des utilisateurs ;
- centrée sur l'architecture du logiciel ;

qui décrit les actions et les informations dans une seule entité.

Étude comparative entre Merise et UML

Merise	UML
Méthode d'analyse et de conception de système d'information	Langage de représentation d'un SI
Méthodes de modélisation de données et traitements orienté bases de données relationnelles	Système de notation orienté objet
relationnel	Objet
Franco-Français	Internationale
Schéma directeur, étude préalable, étude détaillée et la réalisation	Langage de modélisation des systèmes standards, qui utilise des diagrammes pour représenter chaque aspect d'un système : statique, dynamique… en s'appuyant sur la notion d'orienté objet

Le langage SQL

Présentation

Le langage SQL (*Structured Query Language*) peut être considéré comme le langage d'accès normalisé aux bases de données. Il est aujourd'hui supporté par la plupart des produits commerciaux que ce soit par les systèmes de gestion de bases de données micro tel que *Access* ou par les produits plus professionnels tels que *Oracle*.

Le succès du langage SQL est dû essentiellement à sa simplicité et au fait qu'il s'appuie sur le schéma conceptuel pour énoncer des requêtes en laissant le SGBD responsable de la stratégie d'exécution. Le langage SQL propose un langage de requêtes ensembliste et assertionnel. Néanmoins, le langage SQL ne possède pas la puissance d'un langage de programmation : entrées/sorties, instructions conditionnelles, boucles et affectations. Pour certains traitements, il est donc nécessaire de coupler le langage SQL avec un langage de programmation plus complet.

De manière synthétique, on peut dire que SQL est un langage relationnel, il manipule donc des tables (*i.e.* des relations, c'est-à-dire des ensembles) par l'intermédiaire de requêtes qui produisent également des tables.

Les catégories d'instructions

Les instructions SQL sont regroupées en catégories en fonction de leur utilité et des entités manipulées. Nous pouvons distinguer cinq catégories, qui permettent :

- la définition des éléments d'une base de données (tables, colonnes, clés, index, contraintes…),
- la manipulation des données (insertion, suppression, modification, extraction…),
- la gestion des droits d'accès aux données (acquisition et révocation des droits),
- la gestion des transactions,
- et enfin le SQL intégré.

Le langage de définitions de données

Le **langage de définition de données** (LDD, ou *Data Definition Language*, soit DDL en anglais) est un langage orienté au niveau de la structure de la base de données. Le LDD permet de créer, modifier, supprimer des objets. Il permet également de définir le domaine des données (nombre, chaîne de caractères, date, booléen…) et d'ajouter des contraintes de valeur sur les

données. Il permet enfin d'autoriser ou d'interdire l'accès aux données et d'activer ou de désactiver l'audit pour un utilisateur donné.

Les instructions du LDD sont :

Instructions du LDD

CREATE

La commande de création de tables la plus simple ne comportera que le nom et le type de chaque colonne de la table. À la création, la table sera vide, mais un certain espace lui sera alloué.

RENAME

Cette commande permet de renommer une colonne ou une table.

ALTER

La syntaxe de déclaration de contrainte est identique à celle vue lors de la création de tables.

Si des données sont déjà présentes dans la table au moment où la contrainte d'intégrité est ajoutée, toutes les lignes doivent vérifier la contrainte. Dans le cas contraire, la contrainte n'est pas posée sur la table.

DROP

Supprimer une table revient à éliminer sa structure et toutes les données qu'elle contient.
Les *index* associés sont également supprimés.

TRUNCATE

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 à pour but de supprimer les données ainsi que la table qui les contient.

Langage de manipulation de données

Le langage de manipulation de données (LMD, ou Data Manipulation Language, soit DML en anglais) est l'ensemble des commandes concernant la manipulation des données dans une base de données. Le LMD permet l'ajout, la suppression et la modification de lignes, la visualisation du contenu des tables et leur verrouillage.

Les instructions du LMD sont : INSERT, UPDATE, DELETE, SELECT, EXPLAIN, PLAN, LOCK TABLE.

Ces éléments doivent être validés par une transaction pour qu'ils soient pris en compte.

Instructions du LMD

INSERT INTO

La commande INSERT permet d'insérer une ligne dans une table en spécifiant les valeurs à insérer. La liste des noms de colonne est optionnelle. Si elle est omise, la liste des colonnes sera par défaut la liste de l'ensemble des colonnes de la table dans l'ordre de la création de la table. Si une liste de colonnes est spécifiée, les colonnes ne figurant pas dans la liste auront la valeur NULL.

UPDATE

La commande UPDATE permet de modifier les valeurs d'une ou plusieurs colonnes, dans une ou plusieurs lignes existantes d'une table. En l'absence d'une clause WHERE, toutes les lignes sont mises à jour.

DELETE

La commande DELETE permet de supprimer des lignes d'une table. Toutes les lignes pour lesquelles « predicat » est évalué à vrai sont supprimées. En l'absence de clause WHERE, toutes les lignes de la table sont supprimées.

Syntaxe de quelques requêtes SQL

```
CREATE TABLE nom_table (nom_col1 TYPE1, nom_col2 TYPE2, ...)
```

```
ALTER TABLE nom_table {ADD/MODIFY} ([nom_colonne type [contrainte], ...])
```

```
ALTER TABLE nom_table ADD [CONSTRAINT nom_contrainte] contrainte
```

```
ALTER TABLE nom_table RENAME COLUMN ancien_nom TO nouveau_nom
```

```
ALTER TABLE nom_table RENAME TO nouveau_nom
```

```
INSERT INTO nom_table(nom_col_1, nom_col_2, ...) VALUES (val_1, val_2, ...)
```

```
INSERT INTO nom_table(nom_col1, nom_col2, ...) SELECT ...
```

```
UPDATE nom_table SET nom_col_1 = {expression_1 | ( SELECT ...) }, nom_col_2 =
{expression_2 | ( SELECT ...) }, ... nom_col_n = {expression_n | ( SELECT ...) } WHERE
predicat
```

```
DELETE FROM nom_table WHERE predicat
```

Langage d'interrogations de données

- SELECT:

La commande SELECT constitue, à elle seule, le langage permettant d'interroger une base de données. Elle permet de :

- sélectionner certaines colonnes d'une table (projection) ;
- sélectionner certaines lignes d'une table en fonction de leur contenu (sélection)
- combiner des informations venant de plusieurs tables (jointure, union, intersection, différence et division)
- combiner entre elles ces différentes opérations.

Une requête (i.e. une interrogation) est une combinaison d'opérations portant sur des tables (relations) et dont le résultat est lui-même une table dont l'existence est éphémère (le temps de la requête).

Syntaxe de Select

Une requête se présente généralement sous la forme :

```
SELECT [ ALL | DISTINCT ] { * | attribut [, ...] } FROM nom_table [, ...] [ WHERE
condition ]
```

- la clause SELECT permet de spécifier les attributs que l'on désire voir apparaître dans le résultat de la requête ; le caractère *étoile* (*) récupère tous les attributs de la table générée par la clause FROM de la requête ;
- la clause FROM spécifie les tables sur lesquelles porte la requête ;
- la clause WHERE, qui est facultative, énonce une condition que doivent respecter les n-uplets sélectionnés.

Par exemple, pour afficher l'ensemble des n-uplets de la table film, vous pouvez utiliser la requête : `SELECT * FROM film`

La projection et les jointures

- Une projection est une instruction permettant de sélectionner un ensemble de colonnes dans une table. Elle s'effectue à l'aide de la commande SELECT et de la clause WHERE.
- 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. Il y a plusieurs méthodes pour associer 2 tables ensemble. Voici dans la page suivante, la liste des différentes techniques qui sont utilisées :

Les types de jointures

INNER JOIN

Cette commande est une jointure interne pour retourner les enregistrements quand la condition est vrai dans les 2 tables. C'est l'une des jointures les plus communes.

CROSS JOIN

Jointure croisée permettant de faire le produit cartésien de 2 tables. En d'autres mots, permet de joindre chaque lignes d'une table avec chaque lignes d'une seconde table. Attention, le nombre de résultats est en général très élevé.

LEFT JOIN

Jointure externe pour retourner tous les enregistrements de la table de gauche (LEFT = gauche) même si la condition n'est pas vérifié dans l'autre table.

RIGHT JOIN

Jointure externe pour retourner tous les enregistrements de la table de droite (RIGHT = droite) même si la condition n'est pas vérifié dans l'autre table.

FULL JOIN

Jointure externe pour retourner les résultats quand la condition est vrai dans au moins une des 2 tables.

SELF JOIN

Cette commande permet d'effectuer une jointure d'une table avec elle-même comme si c'était une autre table.

Une sous requête

Dans le langage SQL une sous-requête (aussi appelé "requête imbriquée" ou "requête en cascade") consiste à exécuter une requête à l'intérieur d'une autre requête. Une requête

imbriquée est souvent utilisée au sein d'une clause WHERE ou de HAVING pou remplacer une ou plusieurs constante.

Il y a plusieurs façons d'utiliser les sous-requêtes. De cette façon il y a plusieurs syntaxes envisageables pour utiliser des requêtes dans des requêtes.

L'exemple ci-dessous est une exemple typique d'une sous-requête qui retourne un seul résultat à la requête principale:

```
SELECT * FROM `table` WHERE `nom_colonne` = ( SELECT `valeur` FROM
`table2` LIMIT 1 )
```

SQL AS

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.

- Alias sur une colonne: qui 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. Exemple: Une colonne qui s'appelle normalement **c_iso_3166** peut être renommée "code_pays" (cf. le code ISO 3166 correspond au code des pays), ce qui est plus simple à comprendre dans le reste du code par un développeur.
- 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:

```
SELECT colonne1 AS c1, colonne2 FROM `table`
```

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

Union et les clauses SQL

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. 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.

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 liste regrouper les ventes par clients identiques et d'obtenir le coût total des achats pour chaque client.

NB: la commande GROUP BY doit toujours s'utiliser après la commande WHERE et avant la commande HAVING.

Syntaxes

```
SELECT * FROM table1 UNION SELECT * FROM table2
```

```
SELECT colonne1, fonction(colonne2) FROM table GROUP BY colonne1
```

```
SELECT colonne1, colonne2 FROM table ORDER BY colonne1
```

```
SELECT colonne1, colonne2, colonne3 FROM table ORDER BY colonne1 DESC, colonne2 ASC
```

```
SELECT colonne1, SUM(colonne2) FROM nom_table GROUP BY colonne1 HAVING fonction(colonne2)
operateur valeur
```

DISTINCT

La commande DISTINCT permet d'éviter les redondances dans les résultats lors de l'utilisation de la commande SELECT. Pour le Système de Gestion de Bases de Données (SGBD) Oracle, cette requête est remplacée par la commande "UNIQUE".

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.

CASE

Dans le langage SQL, la commande "CASE … WHEN …" permet d'utiliser des conditions de type "si / sinon" (cf. if / else) similaire à un langage de programmation pour retourner un résultat disponible entre plusieurs possibilités.

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 déterminés. 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.

LIMIT

La clause LIMIT est à utiliser dans une requête SQL pour spécifier le nombre maximum de résultats que l'ont souhaite obtenir. Cette clause est souvent associé à un OFFSET, c'est-à-dire effectuer un décalage sur le jeu de résultat.

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.

Syntaxes

```
SELECT DISTINCT ma_colonne FROM nom_du_tableau
```

```
CASE a WHEN 1 THEN 'un' WHEN 2 THEN 'deux’ WHEN 3 THEN 'trois'   ELSE 'autre' END
```

```
SELECT * FROM table LIMIT 10
```

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

```
SELECT nom_colonne FROM table WHERE nom_colonne IN ( valeur1, valeur2, valeur3, ... )
```

```
SELECT * FROM table WHERE nom_colonne BETWEEN 'valeur1' AND 'valeur2'
```

Les fonctions d’agrégations

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 ()

Elle 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.

COUNT ()

Elle 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

SUM ()

Elle 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.

MAX ()

Elle 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.

Syntaxes

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

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

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

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

```
SELECT colonne1, MAX(colonne2) FROM table GROUP BY colonne1
```

```
SELECT colonne1, MIN(colonne2) FROM table GROUP BY colonne1
```

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

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

Conclusion

Apparues il y a de nombreuses années, les bases de données ne sont pas près de disparaitre. Elles deviennent omniprésentes, et dans tous les domaines.

Elles servent à stocker des volumes de données en constante progression et de manière exponentielle, leurs usages sont de plus en plus variés, il en va de même pour les outils servant à les exploiter. Tous les types de données sont stockés, des informations utilisateurs, des images en passant par des informations de cartographie ou au morphing pour la reconnaissance faciale. L'évolution rapide du web et des applications mobiles génère des quantités de nouvelles données, et a amené un besoin pour des moteurs de stockage très simples et faciles à mettre en œuvre.

Toute cette masse d'informations n'est pas seulement stockée, mais elle est traitée afin d'obtenir des indicateurs et des tendances, c'est à ce moment que l'informatique décisionnelle et le big data rentrent en jeu.