MEC 1310 : TI en Génie Mécanique

MODULE F : SYSTÈME DE GESTION DE BASES DE DONNÉES

RICARDO CAMARERO
Département de génie mécanique
École Polytechnique de Montréal

Mars 2011

ÉCOLE
POLYTECHNIQUE
MONTRÉAL

Table des matières

Chapitre 1

Système de Gestion de Bases de Données

Un **système informationnel** est défini comme un ensemble de ressources comprenant des personnes qui exécutent des processus, sur du matériel informatique avec des logiciels utilisant des bases de données destinées à la gestion de l'information au sein d'une organisation ou d'un projet. Il est important de souligner que le système d'information n'est pas uniquement la base de données, mais englobe également tout le processus du cheminement et de traitement de ces données dans une entreprise, et que de tels systèmes existent dans le but spécifique de transformer ces données en information.

Une **base de données**[1] (BD), est définie comme un ensemble d'informations sur un sujet qui est exhaustif, non redondant, structuré et persistant. Puisque les données doivent habituellement être rendues disponibles à plusieurs utilisateurs, la notion de base de données est généralement couplée à celle de réseau, afin de mettre en commun ces informations, d'où le nom de base.

1.1 Modèle conceptuel d'un SGBD

Afin de pouvoir contrôler les données ainsi que leur accès par les utilisateurs, le besoin d'un système de gestion s'est vite fait ressentir. Aujourd'hui, la gestion des bases de données se fait grâce à un **Système de Gestion de Bases de Données**[2] (SGBD), qui peut être défini comme un logiciel permettant de décrire, modifier, interroger et administrer les données de plusieurs bases de données, ainsi que d'en gérer l'accès par leurs utilisateurs. Pour assurer l'atteinte de ces

[1] «DataBase» (DB) en anglais

[2] «DataBase Management System» (DBMS) en anglais

objectifs (surtout les deux premiers), trois niveaux de description des données ont été définis par la norme ANSI/SPARC, soit :

1. **Niveau interne :** description du stockage des données au niveau des unités de stockage, des fichiers, ... On appelle cette description le schéma interne.
2. **Niveau conceptuel :** description de la structure de toutes les données qui existent dans la base, description de leurs propriétés (relations qui existent entre elles) c'est-à-dire de leur sémantique inhérente, sans soucis d'implémentation physique ni de la façon dont chaque groupe de travail voudra s'en servir. On appelle cette description le schéma conceptuel.
3. **Niveau externe :** description pour chaque utilisateur de sa perception des données. On appelle cette description le schéma externe ou vue.

Cette norme à trois niveaux permet d'avoir une indépendance entre les données et le traitement. En séparant les considérations informatiques des aspects conceptuels, on permet aux usagers non-informaticiens de participer à la conception et à l'utilisation de la base de données. D'une manière générale un SGBD doit avoir les caractéristiques suivantes :

Indépendance physique : le niveau physique peut être modifié indépendamment du niveau conceptuel. Cela signifie que tous les aspects matériels de la base de données n'apparaissent pas pour l'utilisateur, il s'agit simplement d'une structure transparente de représentation des informations.

Indépendance logique : le niveau conceptuel doit pouvoir être modifié sans remettre en cause le niveau physique, c'est-à-dire que l'administrateur de la base doit pouvoir la faire évoluer sans que cela gêne les utilisateurs.

Manipulation : des personnes ne connaissant pas la base de données doivent être capables de décrire leurs requêtes sans faire référence à des éléments techniques de la base de données.

Rapidité des accès : le système doit pouvoir fournir les réponses aux requêtes le plus rapidement possible, cela implique des algorithmes de recherche rapides.

Administration centralisée : le SGBD doit permettre à l'administrateur de pouvoir manipuler les données, insérer des éléments et vérifier son intégrité de façon centralisée.

Limitation de la redondance : le SGBD doit pouvoir éviter dans la mesure du possible des informations redondantes, afin d'éviter d'une part un gaspillage d'espace mémoire mais aussi des erreurs.

Vérification de l'intégrité : les données doivent être cohérentes, et en plus, lorsque des éléments font référence à d'autres, ces derniers doivent nécessairement être présents.

Partage des données : le SGBD doit permettre l'accès simultané à la base de données par plusieurs utilisateurs.

Sécurité des données : le SGBD doit présenter des mécanismes permettant de gérer les droits d'accès aux données selon les utilisateurs.

On ne parle plus ici de système de BD, mais bien de systèmes de **gestion** de BD, puisque les SGBD sont non seulement capables de gérer plusieurs BD, mais aussi de gérer les utilisateurs de celles-ci. En pratique, les SGBD contiennent une méta-BD permettant l'administration d'autres BD, ainsi que de leurs utilisateurs, d'où le nom de SGBD.

En raison de leur grande flexibilité, les SGBD sont devenus l'outil principal de gestion des systèmes informationnels des entreprises et organisations diverses. Que se soit pour la gestion des prêts de livres d'une bibliothèque, la gestion du personnel et de la paie, la gestion de l'inventaire et des achats, la gestion des ventes et de la facturation, la gestion des opérations comptables, la gestion des dossiers étudiants d'un établissement d'enseignement, la gestion des horaires du personnel d'un hôpital, la facturation des transactions par cartes de crédits, la facturation des communications cellulaires, la gestion des réservations de sièges d'une compagnie aérienne, les SGBD peuvent s'acquitter de la tâche en toute sécurité.

L'ingénieur mécanique doit aussi maîtriser cet outil afin de gérer les systèmes informationnels techniques rencontrés dans l'exercice de sa profession. Par exemple, un système de gestion de maintenance préventive, un système de gestion de l'outillage de fabrication, un système de gestion des pièces de produits (dans le domaine de l'automobile par exemple) ou tout autre système de gestion d'informations techniques d'entreprises d'ingénierie.

Parmi les principaux SGBD commerciaux sur ordinateurs de moyenne et grande puissance, on trouve les systèmes suivants :

- Sybase à www.sybase.com ;
- Oracle à www.oracle.com ;
- DB2 à www.ibm.com ;
- SQL Server à www.microsoft.com.

Les éditeurs suivants proposent les principaux SGBD sur micro-ordinateurs :

- Paradox, FoxPro, FileMaker ;
- Access à www.microsoft.com.

Les principaux SGBD en logiciel libre sont :
– FirebirdSQL à www.firebirdsql.org ;
– PostgreSQL à www.postgresql.ord ;
– MySQL à www.mysql.com .

1.2 Architecture d'accès

Afin d'atteindre l'objectif de partage des données entre plusieurs utilisateurs, les SGDB peuvent être déployés selon une **architecture d'accès local** ou une **architecture d'accès client/serveur**. Lorsque la BD réside sur le disque d'un poste utilisateur ou sur le disque d'un serveur de fichiers, l'architecture d'accès est dite locale, et ceci, même si le fichier de BD est partagé avec d'autres postes utilisateurs, comme le montre la Figure 1.1. Il est important de noter que le logiciel SGBD charge une copie de la BD dans la mémoire du poste utilisateur et s'exécute entièrement sur le poste de cet utilisateur. Durant ce temps, les autres utilisateurs ne peuvent qu'ouvrir la BD en lecture seulement, et ne verront donc pas les mises à jour effectuées par le premier utilisateur, à moins que celui-ci effectue une sauvegarde et que les autres utilisateurs rechargent une nouvelle copie de la BD.

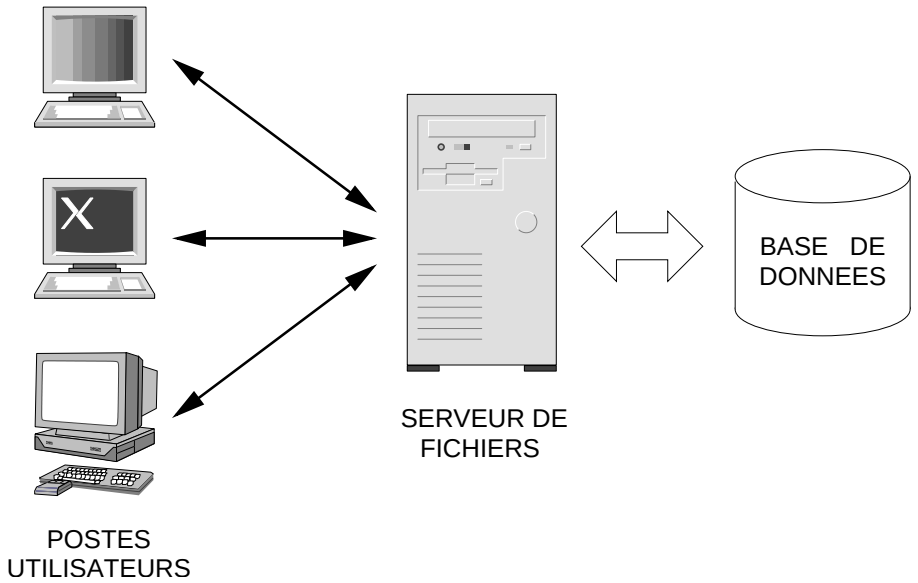


FIG. 1.1 – Architecture d'accès local

Afin d'éviter ce problème de partage non réciproque entre plusieurs utilisateurs, la majorité des SGBD reposent sur une architecture d'accès dite client/-serveur. Dans cette architecture, les données ne sont pas stockées sur le disque d'un serveur de fichiers, ni sur le disque d'un poste utilisateur (appelé client), mais plutôt sur le disque d'un poste spécialement dédié à la base de données (appelé serveur SGBD). Cette architecture fonctionne selon le schéma de la Figure 1.2 :

- le client émet une requête acheminée vers le serveur grâce à son adresse et un numéro de port qui désigne le service particulier requis ;
- le serveur reçoit la demande et répond à l'aide de l'adresse du poste client et au port correspondant.

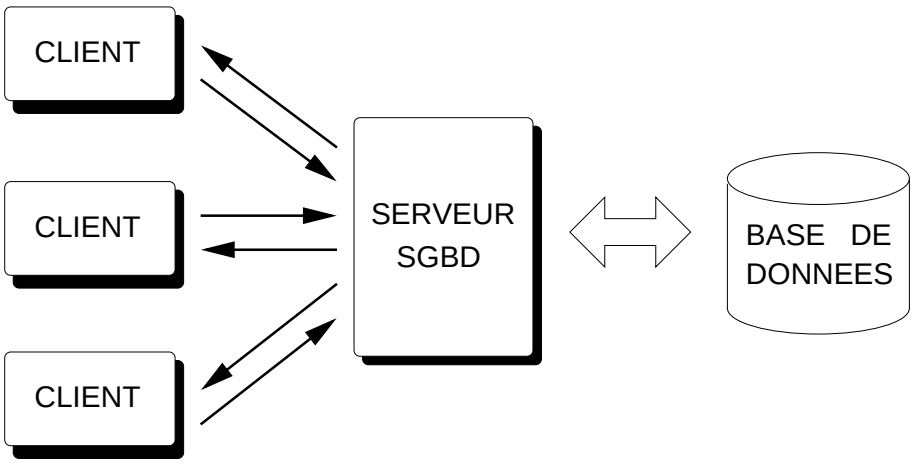


FIG. 1.2 – Architecture d'accès client/serveur

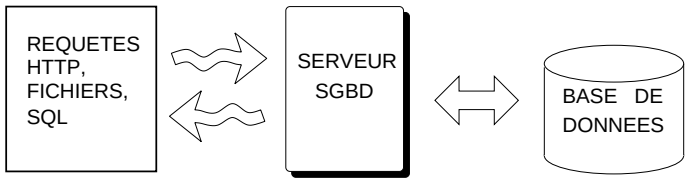
Ainsi, le poste serveur doit *servir* les postes clients en répondant à toutes leurs requêtes d'ajout, de modification ou de consultation des données des différentes bases de données qu'il contient. L'architecture d'accès client/serveur est particulièrement recommandée pour les SGBD nécessitant un haut niveau de fiabilité. Par rapport à une architecture d'accès local, ses principaux atouts sont :

- **des ressources centralisées** : étant donné que le serveur est au centre du réseau, il peut gérer des ressources communes à tous les utilisateurs, comme par exemple une base de données centralisée, afin d'éviter les problèmes de redondance et contradiction ;
- **une meilleure sécurité** : car le nombre de points d'entrée permettant l'accès aux données est moins important ;
- **une administration au niveau serveur** : les postes clients ayant peu d'importance dans ce modèle, ils ont moins besoin d'être administrés. Le niveau

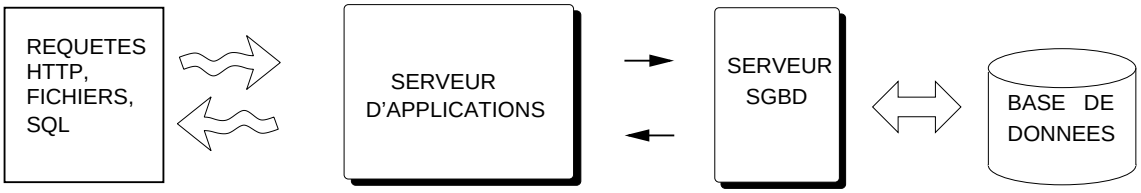
de leur gestion est plus léger.
- **un réseau évolutif** : grâce à cette architecture il est possible de supprimer ou d'ajouter des clients sans perturber le fonctionnement du réseau et sans qu'il soit nécessaire d'y apporter des modifications majeures.

L'architecture d'accès client/serveur a cependant certaines lacunes, parmi lesquelles :
- **un coût plus élevé** : dû à la complexité technique du serveur ;
- **un maillon faible** : le serveur est le seul maillon faible de l'architecture client-serveur, étant donné que tout le réseau est structuré autour de lui ! Heureusement, le serveur a une grande tolérance aux pannes (notamment grâce à des systèmes de disque tels que RAID).

L'architecture d'accès client/serveur est habituellement déployée selon une configuration à deux ou trois niveaux. La configuration à deux niveaux[3] caractérise les systèmes dans lesquels le client demande une ressource et le serveur lui répond directement, comme le montre la Figure 1.3(a). Le traitement des données ainsi échangées se fait par des applications logicielles sur le poste client, ce qui se traduit par un traffic élevé sur le réseau.



(a) Configuration à 2 niveaux



(b) Configuration à 3 niveaux

FIG. 1.3 – Configurations de l'architecture client/serveur

La configuration à trois niveaux (appelée « 3-tier ») requiert l'utilisation d'un serveur d'application entre le client et le serveur SGBD (voir Figure 1.3(b)). Ainsi, les fonctions sont :
- **le client** : le demandeur de ressources ;

[3] (aussi appelée « 2-tier », « tier » signifiant étage en anglais)

- **le serveur d'application** (généralement un serveur Web) : le serveur chargé de fournir la ressource et chargé du traitement des données au lieu du poste client, mais faisant appel à un autre serveur ;
- **le serveur secondaire** (généralement un serveur de base de données), fournissant un service au premier serveur.

La configuration à deux niveaux est une architecture client/serveur dans laquelle le serveur est polyvalent, c'est-à-dire qu'il est capable de fournir directement l'ensemble des ressources demandées par le client. Dans la configuration à trois niveaux par contre, les applications sont réparties entre les deux serveurs chacun spécialisé dans une tâche (serveur Web versus serveur SGBD). Ainsi, la configuration à trois niveaux permet :

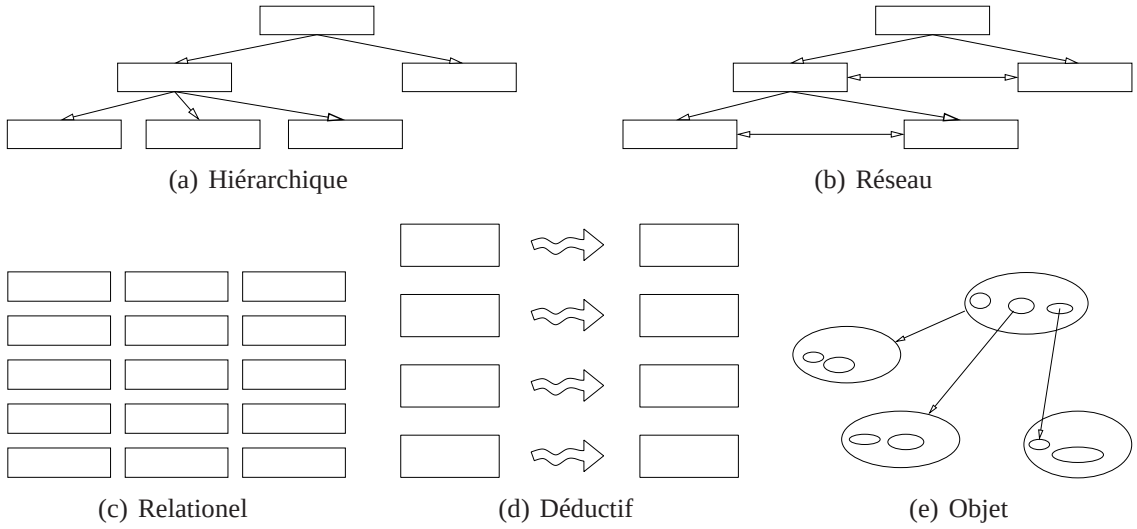
- une plus grande flexibilité et souplesse ;
- une plus grande sécurité (la sécurité peut être définie pour chaque service) ;
- de meilleures performances (les tâches sont partagées).

1.3 Modèles de données

Il existe cinq modèles pouvant être utilisés pour structurer les données stockées dans une BD. Cependant, chaque SGBD est conçu pour ne traiter qu'un seul de ces modèles de données (voir Figure 1.4) :

- **le modèle hiérarchique** : les données sont classées hiérarchiquement, selon une arborescence descendante. Ce modèle utilise des pointeurs entre les différents enregistrements. Il s'agit du premier modèle de SGBD.
- **le modèle réseau** : Comme le modèle hiérarchique, ce modèle utilise des pointeurs vers des enregistrements. Toutefois la structure n'est plus forcément arborescente dans le sens descendant.
- **le modèle relationnel** : les données sont enregistrées dans des tableaux à deux dimensions (lignes et colonnes). La manipulation de ces données se fait selon la théorie mathématique des relations.
- **le modèle déductif** : les données sont représentées sous forme de table, mais leur manipulation se fait par calcul de prédicats.
- **le modèle objet** : les données sont stockées sous forme d'objets, c'est-à-dire de structures appelées classes présentant des données membres. Les champs sont des instances de ces classes.

Le premier système de BD a été conçu pour la gestion des données du programme Apollo de la NASA. Les données étaient structurées dans des hiérarchies, d'une manière semblable à l'organisation des répertoires dans un système



(a) Hiérarchique (b) Réseau

(c) Relationel (d) Déductif (e) Objet

FIG. 1.4 – Modèles de données

de fichiers sur un micro-ordinateur d'aujourd'hui. Les fichiers XHTML d'ailleurs sont une réminiscence moderne des BD hiérarchiques. Mais certains problèmes de stockage ont mené Charles W. Bachman à utiliser (et donc inventer) des BD de type réseaux.

En 1970 au moment où les BD de modèles hiérarchiques et réseaux étaient en plein développement, Edgar F. Codd publie un article où il propose de stocker des données hétérogènes dans des tables, permettant d'établir des relations entre elles. En 1971, il récidive en publiant l'algèbre relationnelle qui est le fondement du traitement de l'information selon le modèle relationnel. De nos jours, ce modèle est extrêmement repondu, mais au début des années 70, cette idée est considérée comme une curiosité universitaire. On doutait alors que les tables ne puissent jamais être générées de manière efficace par un ordinateur. Ce scepticisme n'a cependant pas empêché Edgar F. Codd de poursuivre ses recherches. Un premier prototype de base de données relationnelles est construit dans les laboratoires d'IBM.

En 1980, les premiers SGBD relationnel apparaissent sur le marché. En 1987, le langage de requête structuré SQL[4] qui étend l'algèbre relationnelle est standardisé. En 1990, les SGBD relationnels ont mûris et dominent le marché, lorsque apparaissent les premiers SGBD orientés objet. À l'heure actuelle, les SGBD

[4] Structured Query Language, en anglais

sont présents dans de nombreux logiciels, sont très utilisés dans les BD, et représentent une industrie de plusieurs milliards de dollars.

1.4 Modèle relationnel de base de données

Le *modèle relationnel de base de données* établit des relations entre des entités (contenues dans des tables) et énonce des contraintes quant à leur contenu et à leur utilisation au sein de la BD. Le modèle relationnel est composé :

- d'une partie *structurante*, qui décrit comment la base de données doit être construite ;
- d'une partie *manipulatoire*, qui décrit quelles sont les opérations possibles sur les données ; et
- d'une partie *intégrité*, qui décrit les règles référentielles permettant de garantir la validité des données.

Selon le modèle relationnel, les tables stockent des données et constituent dès lors des blocs essentiels de n'importe quelle BD relationnelle. Ce stockage des données est également effectué sans que l'on ait à se soucier de leur stockage physique dans un ou des fichiers sur les disques du serveur SGBD.

1.4.1 Une situation : La gestion d'un atelier d'usinage

On propose une situation concrête qui servira d'exemple pour la présentation des concepts et pour illustrer la méthodologie de conception, création et d'exploitation d'une base de données. Il s'agit du système d'information d'un atelier d'usinage qui comprend des techniciens, diverses machines outils et une ligne de produits. On veut transformer le processus manuel de la gestion des travaux d'usinage de ce fabriquant de pièces mécaniques par une approche informatique axée sur l'utilisation d'une base de données. Bien que simple, cet exemple permettra d'illustrer la démarche à suivre dans l'application d'un SGBD.

Dans un premier temps on doit modéliser le contenu de la base de données. Une analyse du problème fait ressortir que l'atelier comprend des ressources (les techniciens et les machines outils) , des produits (les pièces) et des processus (travaux d'usinage, facturation...). Ensuite, on doit identifier les liens et les interactions entre ces diverses entités.

1.4.2 Les tables

Dans une base de données, les informations sont contenues dans des tables. Une **table** est définie comme une collection d'enregistrements sous un thème commun. Dans l'exemple de l'atelier d'usinage, la table ***Operateurs*** est un ensemble d'informations (thème) qui charactérise les opérateurs, par exemple les noms et taux horaires etc. de ces techniciens. Bien que chaque opérateur soit toujours jumelé à une et une seule machine (l'opérateur numéro *i* est toujours jumelé à la machine *i*), les informations sur les opérateurs et les machines sont stockées dans deux tables séparées parce qu'elles constituent des thèmes(ou type de données) différents. Un **enregistrement** [5], est défini comme un regroupement

NoOperateur	NomOperateur	TauxOperateur
1	Guy Bélanger	35
2	Bernard StPierre	40
3	Annie Guay	38

TAB. 1.1 – Table des opérateurs

NoMachine	NomMachine	TauxMachine
1	Matsura 5 axes	125
2	Huron KX8 UGV	165
3	Mazak BZ-4	85

TAB. 1.2 – Table des machines

de plusieurs champs permettant de décrire un seul élément ou événement dans une table. Chaque enregistrement de la table ***Pieces*** décrit l'élément pièce à fabriquer par son numéro de pièce et le temps d'usinage requis pour sa fabrication. On peut imaginer d'autres informations concernant le modèle CAO, le matériau etc. De même, chaque enregistrement de la table ***Travaux*** décrit l'événement d'émission d'un bon de travail, c'est-à-dire l'assignation d'une machine par son numéro, pour la fabrication d'une pièce de numéro donné, en une quantité requise. Un enregistrement est composé de plusieurs champs. Un champ [6] est défini comme *un objet qui est caractérisé par un type et un domaine prédéfini*.

[5] « record » en anglais
[6] « field » en anglais

NoPiece	NomPiece	TempsPiece
1	F178-12	10
2	F200-14	15
3	G130-29	24
4	H200-12	16

TAB. 1.3 – Table des pièces

NoTravail	NoMachine	NoPiece	QuantitePiece
1	2	4	10
2	1	2	25
3	3	1	4
4	2	3	16

TAB. 1.4 – Table des travaux

Par exemple, la 1ère colonne de la table ***Machines*** est de type *NuméroAuto*, c'est-à-dire un entier positif de 4 octets avec une incrémentation automatique lors de l'ajout de données. En plus de son type, un champs est caractérisé par son domaine qui indique la plage de valeurs admissibles, dans cet exemple, *toutes valeurs entières positives*. Afin de garantir l'intégrité de la base de données, le SGBD s'assure que les données entrées dans un champ sont comprises dans le domaine prédéfini pour celui-ci. Un champ est la plus petite unité de données que peut manipuler un SGBD.

Ainsi, la 2ème colonne de la table ***Operateurs*** contient le nom des opérateurs. Ce champ est de type *chaîne de caractères de longueur variable* et de domaine maximal de 30 caractères. Ce champ est utilisé pour stocker à la fois, les noms et prénoms des opérateurs, ainsi il ne sera pas possible au SGBD de gérer séparément ces deux informations par la suite. Si l'on désire gérer les noms et prénoms des opérateurs, il est nécessaire de fractionner ce champ en deux.

En résumé, une table est un tableau (dans le sens informatique) où les données sont stockées sous la forme de colonnes et de rangées. Les colonnes comprennent des données d'un même type, tandis que les rangées (enregistrements) réunissent des données diverses décrivant un élément de l'entité représentée par la table.

1.4.3 Indexation

Une **clé primaire** [7] est définie comme un champ constituant un identifiant unique pour les enregistrements de la table. Une clé primaire doit obligatoirement être définie pour chacune des tables. À la Figure 1.5, la clé primaire de chaque table apparaît en caractères gras. Pour former cet identificateur, le SGBD a souvent besoin d'un numéro de référence, d'un code produit ou d'un matricule employé afin de garantir cette unicité (Voir les Tableaux 1.1 à 1.4). Souvent, plusieurs champs d'une table sont concaténés pour former cet identifiant ou clé primaire.

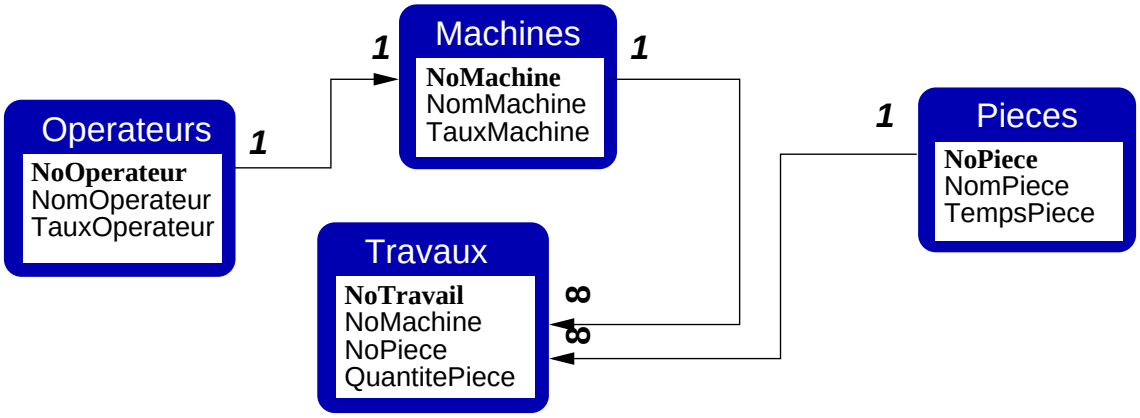


FIG. 1.5 – Schéma relationnel de la BD ***Atelier***

Une **clé étrangère** [8] est définie comme *un ou plusieurs champs d'une table destinés à correspondre avec la clé primaire d'une autre table*. C'est par ce mécanisme que l'on établit des relations entre différentes entités dans une BD. Le champ ***NoPiece*** des tables ***Travaux*** et ***Pieces*** sont, respectivement, des clés étrangère et primaire permettant de relier un bon de travail à un numéro de pièce à fabriquer. Ainsi, le bon de travail no. 1 indique que la machine no. 2 doit fabriquer 10 exemplaires de la pièce no. 4, alors que le bon de travail no. 2 indique que la machine no.1 doit fabriquer 25 exemplaires de la pièce no. 2.

[7] « primary key » en anglais
[8] « foreign key » en anglais

1.4.4 Les relations

Une **base de données relationnelle**[9] est définie comme *une collection de tables reliées entre elles par des relations et ayant un objectif spécifique*. Par exemple, la base de données présentée à la Section 1.4.1, appelée ***Atelier***, est constituée des tables : ***Operateurs***, ***Machines***, ***Travaux*** et ***Pieces***. À l'aide de relations de champs communs, on peut relier certaines entités (c-à-d des tables) de cette BD entre elles pour indiquer des liens spécifiques, tel que le montre la Figure 1.5. Une **relation** est définie comme un ou plusieurs champs communs entre deux tables. Les tables ***Travaux*** et ***Pieces*** possèdent le champ ***NoPiece*** en commun, ces deux tables sont donc liées par une relation entre ces champs.

L'intégrité référentielle est un principe qui assure que les champs communs sont de mêmes type et domaine, et que les données déjà présentes sont compatibles avec la relation. Lors de la création d'une relation, le SGBD vérifie que ce principe est respecté en validant le type et le domaine des données dans ces champs. Par la suite, il ne permettra pas d'entrer un enregistrement ou une modification dans la table ***Travaux*** avec un numéro de pièce qui n'existe pas déjà dans la table ***Pieces***. Ainsi, nous sommes toujours assurés que les bons de travail contiennent un numéro de pièce valide.

Les tables ***Machines*** et ***Travaux*** sont aussi reliées par le champ commun ***NoMachine***. Ces deux relations permettent d'éviter de dédoubler inutilement les données des tables ***Pieces*** et ***Machines*** dans la table ***Travaux***. Ainsi, cette dernière ne contiendra pas directement le temps d'usinage d'une pièce, ni le taux horaire de la machine, mais contiendra plutôt une référence vers la pièce à fabriquer dans la table ***Pieces*** et une référence vers la description de la machine outil dans la table ***Machines***, ainsi que la quantité de pièces à fabriquer.

La relation ***NoMachine*** entre les tables ***Machines*** et ***Travaux*** est de type *1 à plusieurs (infini)*, c'est-à-dire qu'un seul enregistrement de la table ***Machines*** peut correspondre plusieurs enregistrements dans la table ***Travaux***. Heureusement, cela signifie qu'une même machine peut recevoir consécutivement plusieurs bons de travail. Cependant, deux enregistrements de la table ***Machines*** ne peuvent jamais avoir le même numéro de machine. La relation ***NoPiece*** entre les tables ***Pieces*** et ***Travaux*** est aussi de type *1 à plusieurs (1 à ∞)*, c'est-à-dire qu'une même pièce peut être l'objet de plusieurs bons de travail, mais une pièce n'a toujours qu'un seul numéro de pièce. Une relation de type *1 à plusieurs relie toujours une clé primaire à une clé étrangère*.

[9] Relational database en anglais

Les champs ***NoOperateur*** et ***NoMachine*** sont reliés par une relation de type 1 à 1, c'est-à-dire qu'un seul enregistrement de la table ***Operateurs*** correspond à un seul enregistrement de la table ***Machines***. Une relation de type *1 à 1 relie toujours deux clés primaires*. Les relations de type *plusieurs à plusieurs (infini à infini)* ne peuvent pas être directement représentées dans les SGBD relationnels, mais nécessitent plutôt l'ajout d'une table supplémentaire et l'utilisation de deux relations 1 à plusieurs. Ces relations ne seront pas étudiées dans cette introduction.

1.4.5 Principes de normalisation

Dans une base de données relationnelles, les tables sont les blocs de construction des bases de données. L'élaboration d'une structure de tables saine constitue une étape primordiale pour la création d'une base de données efficace et facile à gérer. Pour ce faire, un processus de normalisation du schéma relationnel selon cinq règles (qualifiées de formes normales) a été publié et standardisé. L'application de ces règles vise à éviter la redondance inutile des données, afin de minimiser l'espace disque utilisé et empêcher la possibilité de créer des incohérences dans les tables. En pratique, on utilise surtout les trois premières formes normales, et ce processus de normalisation requiert souvent de scinder une table en deux ou de dédoubler un champ servant de clé primaire. Même si le dédoublement d'une clé primaire peut occuper plus d'espace mémoire, l'usager n'a pas de données supplémentaires à saisir et la base de données normalisée devient plus facile à gérer par le SGBD.

Première forme normale (FN1) : éviter qu'un même champ ne contienne plus d'une valeur ou éviter la répétition de champs pour contenir un nombre variable de valeurs par enregistrement.

Prenons, par exemple, le SGBD ***Atelier*** de notre fabricant de pièces qui désire modifier les bons de travail afin de permettre de consigner la fabrication de plusieurs numéros de pièces sur un même bon de travail. La Figure 1.5 montre une modification possible de la table ***Travaux*** qui permet d'atteindre la fonction désirée. Cependant, la table contient plusieurs valeurs dans les champs ***NoMachine***, ***NoPiece*** et **QuantitePiece**, ce qui est interdit par FN1.

La Figure 1.6 montre une deuxième modification possible de la table ***Travaux*** qui permet aussi d'atteindre partiellement la fonction désirée (A, B et C seulement !). Cependant, la table ***Travaux*** contient une répétition des

NoTravail	NoMachine	NoPiece	QuantitePiece
1	3,1	4,2	10,25
2	2,1,3	1,4,2	15,30,10
3	2	3	45

TAB. 1.5 – Schéma relationnel à plusieurs variables par champs non FN1

champs ***NoMachine***, ***NoPiece*** et **QuantitePiece** afin de contenir un nombre variable de valeurs, ce qui est interdit par FN1.

NoTravail	NoMachineA	NoPieceA	QuantitePieceA	NoMachineB	NoPieceB	QuantitePieceB
1	3	4	10	1	2	25
2	2	1	15	1	4	30
3	2	3	45	0	0	0

TAB. 1.6 – Schéma relationnel avec une répétition de champs non FN1

Pour résoudre ce problème en conformité avec la FN1, il faut créer une nouvelle table, appelée par exemple **Lignes_Travaux**, permettant de relier les tables ***Travaux*** (modifiée) et ***Pieces*** en ne consignant qu'une seule pièce par machine par enregistrement, comme le montre le Tableau 1.7. Le nouveau schéma relationnel découlant de ces modifications est illustré à la Figure 1.6.

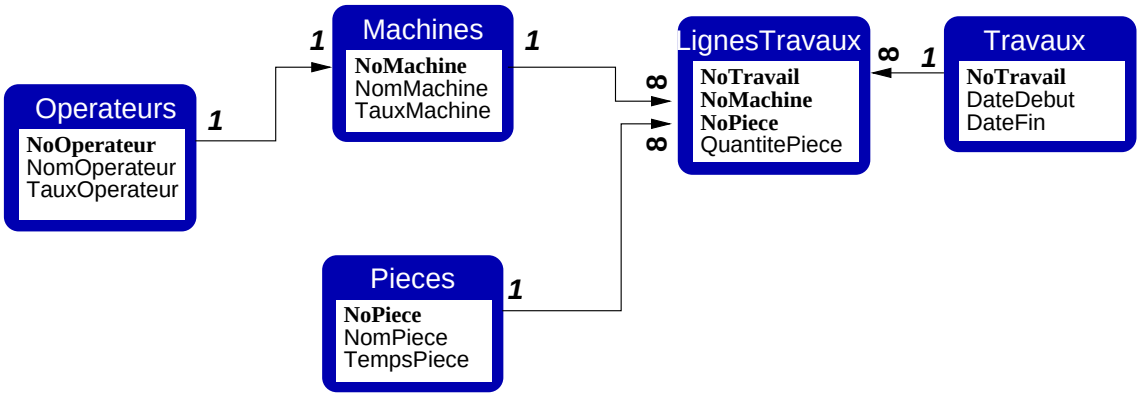


FIG. 1.6 – Schéma relationnel selon FN1 de la BD Atelier modifiée

Deuxième forme normale (FN2) : gestion des relations plusieurs à plusieurs, que nous n'étudierons pas dans cette introduction aux SGBD.

NoTravail	NoMachine	NoPiece	QuantitePiece
1	3	4	10
1	1	2	25
2	2	1	15
2	1	4	30
2	3	2	10
3	2	3	45

NoTravail	DateDebut	DateFin
1	2005-11-21	2006-02-12
2	2005-12-15	2006-01-28
3	2006-014-2	

TAB. 1.7 – Nouvelles Tables des Travaux et Lignes_Travaux

Troisième forme normale (FN3) : éviter que les enregistrements d'une table aient des champs de natures différentes.

Dans notre exemple, un concepteur inexpérimenté aurait très bien pu regrouper les tables **Operateurs** et ***Machines*** en une seule table appelée **Machines_Operateurs**, tel que montrée à la Table 1.8. Clairement, cette table ne contient pas de valeurs redondantes. Cependant, on reconnaît que les opérateurs et les machines sont deux entités de natures différentes. Selon FN3, il faut donc les traiter dans des tables différentes. Le résultat sera deux tables reliées par une relation 1 à 1 entre leurs clés primaires, tel que nous l'avions déjà présenté à la Figure 1.5. Cette forme n'est pas toujours appliquée, puisqu'elle ne permet que de faciliter le traitement futur de l'information.

NoMachine	NomMachine	TauxMachine	NomOperateur	TauxOperateur
1	Matsura 5 axes	125	Guy Bélanger	35
2	Huron KX8 UGV	165	Bernard StPierre	40
3	Mazak BZ-4	85	Annie Guay	38

TAB. 1.8 – Table Machines_Operateurs non FN3

Les autres formes normales ne seront pas étudiées dans ce document d'introduction. Le lecteur intéressé est référé à

$$http://fr.wikipedia.org/wiki/Formes\_normales$$

pour plus d'informations sur ce sujet.

1.5 Nomenclature des objets

La conception efficace d'une base de données passe aussi par une nomenclature signifiante et structurée de ses objets. Le concepteur devrait suivre les règles suivantes :

- Éviter les accents, les espaces ou autres caractères spéciaux dans les noms des tables et champs ou autres objets, et ce, même si un logiciel particulier l'accepte. Par exemple, **Access** est très permissif en ce sens, mais ce n'est pas nécessairement le cas des autres SGBD.
- Utiliser le soulignement « _ » pour assembler des mots sans espace (ex : **Contacts_Telephone**), ou bien intercaler une majuscule (ex : **ContactsTelephone**).
- Nommer les tables, champs et les autres objets avec des noms (identifiants) courts et explicites (ex : **Clients**).
- Conserver les mêmes noms de champs pour les clés primaires dans les clés étrangères. Par exemple, les champs ***NoTravail***, ***NoMachine*** et ***NoPiece*** sont des clés étrangères dans la table **Lignes_Travaux** et des clés primaires dans leurs tables respectives.
- Développer un système de nomenclature où le même libellé est utilisé pour les champs ayant les mêmes fonctions. Par exemple, **NoMachines** et **NoPieces** ont le préfixe **No** parce qu'ils ont la même fonction de numérotation.

Chapitre 2

MySQL

MySQL est un sytème de gestion de bases de données relationnel issu du monde du logiciel libre (Open source) dans la lignée des environnements sous Linux (sytèmes d'exploitation), Apache (serveur Web), gcc (compilateur C) etc. Ce SGBD est disponible pour une multitude de systèmes et totalement gratuit pour des utilisations non-commerciales sous Unix. Sa diffusion auprès d'une large base d'usagers découle de plusieurs caractéristiques qui en font un excellent compromis entre la puissance de traitement, la facilité d'utilisation et la fonctionalité.

Un SGBD est un outil informatique pour la gestion de bases de données de diverses sources et dans diverses applications. L'utilisation d'un système de base de données se justifie lorsque la quantité de données à traiter devient trop volumineuse ou trop complexe pour être traitée manuellement. Il devient alors avantageux de gérer électroniquement les nombreux enregistrements, ce qui permet :

- une réduction du temps de saisie des données ;
- un accès rapide aux enregistrements ;
- des manipulations variées et faciles pour l'extraction d'informations ;
- un accès à distance et électronique dans un environnement Web.

2.1 Contexte

Dans ce chapitre, on présente les concepts du SGBD MySQL et du langage SQL dans une optique d'un usager non-informaticien. La démarche utilisée est basée sur des situations concrêtes qui serviront d'exemples pour la présentation des concepts et de la méthodologie de conception, création et d'exploitation d'une

base de données.

Une situation : La gestion d'un carnet de notes On veut transformer le processus manuel de la gestion de la notation des étudiants dans un cours, par une approche informatique basée sur l'utilisation d'une base de données. Le cours comprend diverses activités qui impliquent de nombreux étudiants et qui donnent lieu à plusieurs types de contrôles dans des situations variées.

Pour chaque type de contrôle, le professeur veut sauvegarder les notes, attribuer des notes selon la nature de l'activité, c-à-d résultat individuel ou en équipe, gérer la remise de certains travaux et les absences. Pour la classe, on veut réaliser certaines analyses (moyennes, histogrammes...), traitement (compilation et pondération) et différents tris selon certains critères.

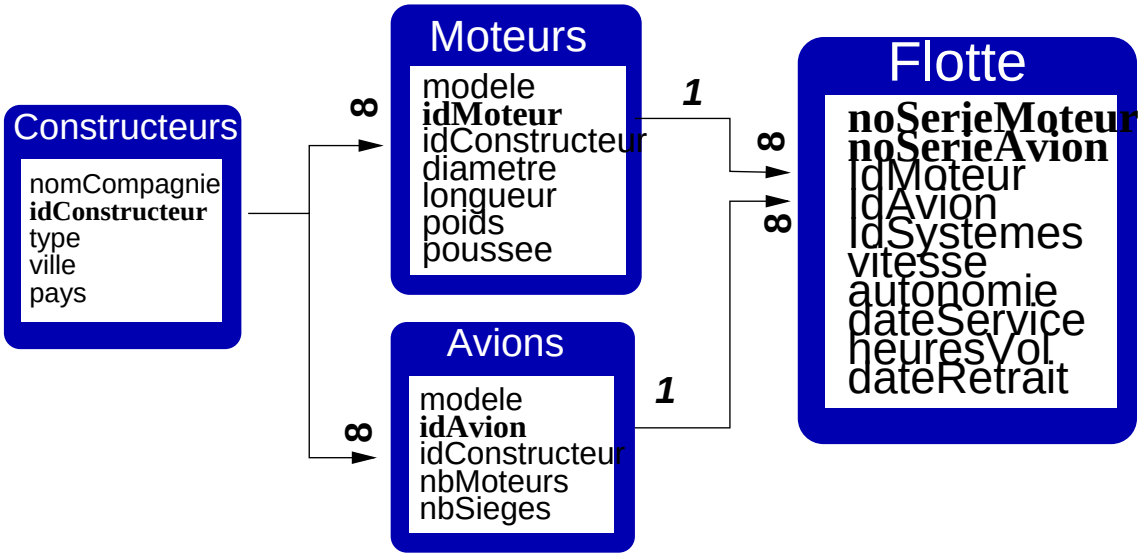


FIG. 2.1 – Schéma relationnel de la BD Aviation

Une situation : La gestion de la flotte d'une société d'aviation

Volontairement simple, ces exemples permettront de montrer la démarche à suivre dans l'application d'un SGBD, et d'illustrer les possibilités qu'offrent ces systèmes. Notament,

1. identifier les objectifs et buts à atteindre ;
2. modéliser le contenu de la base de données.
3. mise en oeuvre et opérationalisation de la BD.

L'étape 1) permet de décrire les résultats que doit renvoyer la base de données et doit précéder la définition précise du contenu. En effet l'organisation des données dépend de l'utilisation qui en sera faite.

2.2 Modèle conceptuel

Un système de gestion de bases de données, comme MySQL, comprend :

une base de données qui indique l'emplacement où sont stockées les données. Celle-ci est constituée de plusieurs tables. Une table est une structure informatique qui organise des données sous forme de rangées et de colonnes. On appelle une rangée un enregistrement et il contient plusieurs champs correspondant aux colonnes de la table.(Voir la Section 1.4.2)

TAB. 2.1 – Table des constructeurs aéronautiques

Nom de la compagnie	idConstructeur	type	ville	pays
Boeing Co.	1	Avionneur	Chicago	USA
Bombardier Aéro.	2	Avionneur	Montréal	Canada
Airbus Industries	3	Avionneur	Toulouse	France
General Electric	4	Motoriste	Schenecdaty	USA
Pratt & Whitney	5	Motoriste	Hartford	USA
Rolls Royce	6	Motoriste	Derby	Angleterre

TAB. 2.2 – Un enregistrement

Bombardier Aéronautique	2	Avionneur	Montréal	Canada

un système de gestion qui regroupe tous les logiciels pour accéder à la base de données, récupérer des enregistrements, y insérer des données, les modifier et les effacer.

un langage qui comprend un ensemble d'instructions pour interagir avec la BD. La communication avec la base de données se fait par le biais d'un langage intitulé SQL (Structured Query Langage) qui est une norme adoptée par l'ensemble des éditeurs de SGBD et intégrée dans leur divers produits.

2.3 Base de données relationnelle

Tel que présenté à la Section 1.4, une base de données comprend plusieurs tables qui correspondent aux différentes entités que l'on veut gérer. Par sytème de BD relationnel, on entend la capacité d'établir des relations ou liens entre les données en provenance de ces différentes tables.

Par exemple, dans la gestion de sa flotte d'appareils un transporteur aérien fait affaires avec plusieurs constructeurs (voir Table 2.1) qui lui fournissent différents appareils et moteurs, ayant chacun ses charactéristiques propres (voir Tables 2.3 et 2.4). Ces quatre tables sont reliées entre elles par des références. Par exemple, dans le premier enregistrement de la table des modèles d'avions, le champ "idConstructeur" fait référence par un index "1" à la table des constructeurs. Ceci se réfère à la clé "1" de la table des constructeurs qui identifie le constructeur "Boeing Co.".

TAB. 2.3 – Table des modèles d'avions

Modèle	idAvion	idConstructeur	nbMoteurs	nbSieges
B747-400	1	1	4	420
B757-200	2	1	2	243
B727	3	1	2	160
A340-600	4	3	2	380
A340-500	5	3	2	313
A330-300	6	3	2	362
A330-200	7	3	2	363
A310-300	8	3	2	259

2.4 Architecture MySQL

L'interaction entre l'usager et les données se fait dans le cadre d'une architecture client-serveur, c-à-d qu'il y a deux programmes qui s'éxécutent, l'un sur le poste de l'usager et l'autre sur la machine où se trouve la base de données (Cela pourrait être le même ordinateur ou bien une machine distante). Le client est un programme qui établit une connexion avec le serveur et lui communique les requêtes de l'usager. Le serveur reçoit, interprète et accède à la base données pour exécuter les requêtes en provenance du client. Le résultat est envoyé au client qui l'affiche sur le poste de l'usager.

TAB. 2.4 – Table des modèles de moteurs

Modèle	idM	idConstructeur	diametre	longueur	poids	poussée
CF34-10D		GE	57	90	3800	18500
CF34-3A		GE	49	103	1625	9220
CF34-8C1		GE	52	128	2350	13790
CF6-50C1		GE	105	183	8966	52500
CF6-80E1A4		G E	106	168	9790	68100
RB211-524G		RR	110	150	9100	58000
RB211-535E4		RR	110	150	9100	40100
TRENT556		RR	97	145	8900	56000
TRENT553		RR	97	145	8900	53000
PW400-100		PW	100	163	9200	64500
JT9D		PW	93	132	8900	56000

TAB. 2.5 – La flotte d'un transporteur

noSerieMoteur	noSerieAvion	idMoteur	idAvion	idSystemes	.....
1358D	S L120-8	1	2		
2945F	pw340-A	2	5		

.....	vitesse	autonomie	dateService	heuresVol	dateRetrait
	890	4500	1982-12-04	2300	2006-10-17
	850	5600	1994-04-12		

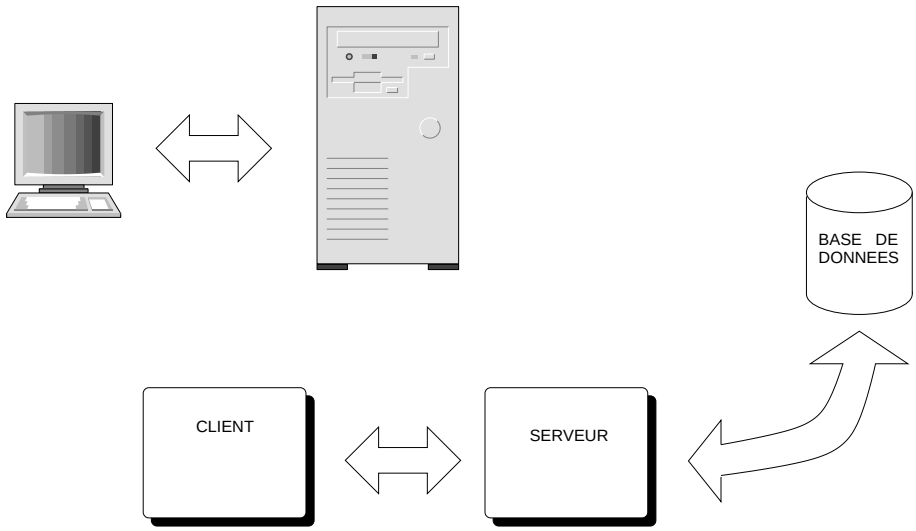


FIG. 2.2 – Architecture client-serveur local

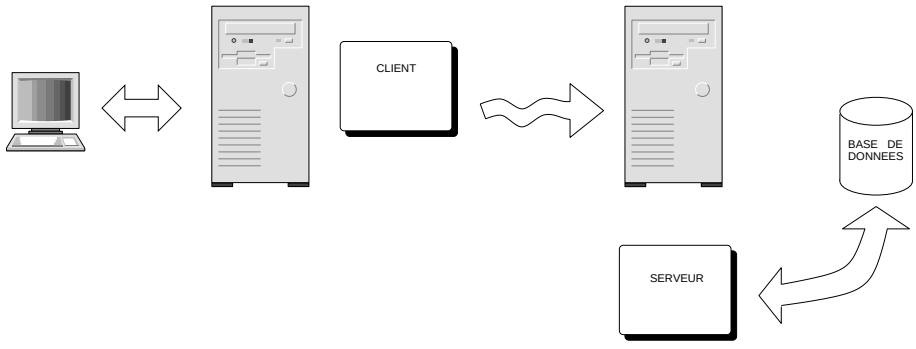


FIG. 2.3 – Architecture client-serveur distant

L'architecture client-serveur offre plusieurs avantages :

usagers multiples : Plusieurs usagers peuvent utiliser la base en même temps. Comme toutes les requêtes passent par le serveur qui gère et contrôle l'accès à une table donnée, les modifications se font dans un ordre déterminé par le serveur.

connexion à distance : Il n'est pas nécessaire pour un usager de se connecter sur la machine où se trouve la base de données. MySQL utilise les protocoles de communication de l'Internet, permettant à un client sur un poste quelconque de fonctionner avec un serveur sur une machine distante. De plus, MySQL intègre un système élaboré de sécurité pour la gestion des usagers et des privilèges d'accès.

2.5 La connexion à la BD

La connexion à la base de données MySQL se fait par la commande **mysql** qui comprend plusieurs options :

mysql -h *host-name* **-u** *user-name* **-p**

-h host-name

Ce paramètre identifie la machine où se trouve la BD sur laquelle l'usager veut se connecter. Si le serveur MySQL est local, c-à-d sur la même machine que l'usager, alors, on peut omettre cette option.

-u user-name

L'identifiant ou nom de l'usager. Sous Unix ce nom peut être le même que celui du compte usager. Alors, on peut omettre cette option et MySQL utilisera automatiquement ce dernier.

-p

Cette option indique au serveur de demander le mot-de-passe de l'usager. Celui-ci pourrait être donné directement sur la ligne de commande, mais sera alors visible à l'écran. Cette pratique est à éviter pour des raisons de sécurité.

L'exemple suivant illustre la connexion de l'usager *tixxx* sur un système Unix avec un serveur MySQL installé en mode local sur son ordinateur :

```
$ mysql -u tixxx -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 76 to server version: 3.23.54

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

L'invite **mysql >** indique que le client est en contact avec le serveur MySQL. Pour se connecter sur une machine distante, il faut préciser l'adresse de cette dernière avec l'option **-h** *host-name* et généralement le nom-usager correspondant. On peut également se connecter au préalable, avec le protocole **ssh** sur une machine distante sur laquelle se trouve un serveur **MYSQL**, et ensuite la connexion avec le SGBD est établie comme suit :

```
$ ssh tixxx@cogito.labos.polymtl.ca
tixxx@cogito.labos.polymtl.ca's password:
[tixxx@cogito tixxx]$ mysql -u tixxx -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 76 to server version: 3.23.54

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

On met fin à une session de travail avec la commande **QUIT** :

mysql > QUIT

```
mysql> QUIT
Bye
```

2.6 La syntaxe des requêtes

Les requêtes s'énnoncent comme des phrases d'un langage naturel, et comprennent un verbe pour décrire l'action souhaitée, suivi d'objects sur lesquels

portera l'action et des qualificatifs (adjectifs et adverbes) pour décrire certains paramètres de la commande. L'ensemble doit respecter une syntaxe propre à SQL (règles d'orthographe et de grammaire) pour être compréhensible par le serveur.

Quelques commandes servent d'indicatif, telles que **NOW()** qui donne l'heure et la date courante :

```
mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2010-10-01 12:11:57 |
+---------------------+
1 row in set (0.02 sec)

mysql>
```

On note que la commande se termine toujours par "**;**", indiquant au serveur que la commande est complète. On peut émettre plusieurs commandes sur une même ligne, en les séparant par une virgule :

```
mysql> SELECT NOW(), USER(), VERSION();
+---------------------+-----------------+-----------+
| NOW()               | USER()          | VERSION() |
+---------------------+-----------------+-----------+
| 2010-10-01 12:12:23 | tixxx@localhost | 3.23.54   |
+---------------------+-----------------+-----------+
1 row in set (0.00 sec)

mysql>
```

Dans ce document, différentes commandes ou les diverses parties d'une même commande seront inscrites sur plusieurs lignes pour en faciliter la lisibilité :

```
mysql> SELECT NOW(),
    -> USER(),
    -> VERSION();
+---------------------+-----------------+-----------+
| NOW()               | USER()          | VERSION() |
+---------------------+-----------------+-----------+
```

```
| 2010-10-01 12:12:56 | tixxx@localhost | 3.23.54   |
+---------------------+-----------------+-----------+
1 row in set (0.00 sec)

mysql>
```

On remarque que l'invite de MySQL a changé de **mysql >** à **- >** , pour montrer que la commande n'est pas complète, et que le serveur est en attente de la suite ou de la fin qui sera indiquée par "**;**".

MySQL est insensible à la casse pour ce qui à trait aux commandes, c-à-d on peut très bien écrire **VERSION()**, ou **version()**. Par contre, il faut respecter les espaces dans le libellé d'une commande. Finalement, la casse dans les noms des bases de données, des tables, ou des champs dans ces dernières doit être scrupuleusement respectée. Par convention, dans ce document les commandes et les options sont inscrites en majuscules et on réserve l'usage des minuscules pour les noms de fichiers et les variables (champs) dans les tables.

Pour faciliter le mode de commande en ligne, deux commandes sont utiles :

Annulation On peut interrompre la saisie d'une commande en cours avec la commande

`\c`

et redonner la main au serveur :

```
mysql> CREATE TABLE avions
    -> (
    -> modele VARCHAR(15) NOT NULL,
    -> idModele INT NOT NULL,
    -> idConstructeur INT(10) NOT NULL,
    -> nombreMoteurs INT NOT NULL,
    -> \c
mysql>
```

Script *.sql Soumettre des requêtes directement sur la ligne de commande peut s'avérer fastidieux et surtout, donner lieu à de multiples possibilités d'erreurs. Une approche plus pratique consiste à inscrire les requêtes dans un fichier avec un éditeur de texte qui se modifie et se corrige facilement. Lorsque le travail de rédaction des requêtes est terminé, on dirige le contenu du fichier ou script vers le serveur qui traite les requêtes qui y sont contenues. On utilise la commande de redirection "**<**" du sytème Unix :

mysql > ma_BD < fichier_de_commande.sql ;

On peut également utiliser la commande **SOURCE** :

mysql > SOURCE fichier_de_commande.sql ;

2.7 Création de la BD

L'utilisation d'une base de données suppose qu'elle existe et que les données qu'elle contient sont organisées dans une structure appropriée, en fonction des besoins en information des usagers. Cette démarche de conception et de mise en oeuvre comprend plusieurs étapes :

1. Modélisation des données et structuration des tables ;
2. Création et initialisation de la base de données ;
3. Génération des données et affectation dans les tables ;
4. Gestion et entretien de la BD.

Après la phase de modélisation et de conception, la mise en oeuvre proprement dite nécessite des instructions de création, **CREATE DATABASE**, et d'accès, **USE**, à la BD.

```
$ mysql -u tixxx -p
Enter password: ******
 Welcome to MySQL
Type "help" for help

mysql > CREATE DATABASE ma_BD ;
mysql >
```

Une fois la BD créée, on y accède, ou on l'active, par la commande **USE** :

```
$ mysql -u tixxx -p  ;
Enter password: ******
 Welcome to MySQL
Type "help" for help
mysql > USE ma_BD;
```

On peut également activer une base de données existante, lors de l'étape de connexion :

```
$ mysql -u tixxx -p ma_BD ;
Enter password: ******
 Welcome to MySQL
Type "help" for help
mysql >
```

2.8 Construction des tables

La construction d'une table comprend sa déclaration, suivi de l'insertion des données dans les champs, c-à-d les enregistrements. Pour illustrer cette étape, l'exemple de la flotte d'une société de transport aérien sera utilisé. Les données sont regroupées en plusieurs tables. De façon générale le choix des tables et de leurs contenus n'est pas unique, mais dépendent de l'application ou des exigences de la situation. Ici, le but est d'illustrer la démarche et de présenter l'utilisation des fonctionalités d'un SGBD.

La table des constructeurs aéronautiques (Tableau 2.1) Elle contient une liste de constructeurs d'avions qui fournissent les appareils de la flotte de la société d'aviation. Les enregistrements comprennent le nom, un numéro d'identification, et une adresse. Cette information comprend le nom de la ville et le pays. À première vue créer une seule colonne qui contiendrait ces deux éléments semble la solution la plus simple. Cependant, avec le nom de la ville en premier il ne sera pas possible de trier selon le pays et vice-versa. C'est pour cette raison que cette donnée est dans deux champs.

La table des modèles d'avions (Tableau 2.3) Cette table contient une liste des différents appareils et leurs charactéristiques principales.

La table de la flotte (Tableau 2.5) Cette liste énumère chacun des appareils de la compagnie avec son numéro de série, son type, le fournisseur, la date de mise en service, les heures de vol et la date de retrait de service.

La création d'une table se fait avec la déclaration :

```
CREATE TABLE nom_table ( champs_description )
```

La syntaxe comprend le nom de la commande (en majuscule) , le nom de la table et, entre parenthèses, la description des champs de l'enregistrement. Pour la table des constructeurs cela donne :

```
CREATE TABLE constructeurs
 (
    nomCompagnie   VARCHAR(35) NOT NULL ,
     idConstructeur   INT UNSIGNED NOT NULL
                      AUTO_INCREMENT PRIMARY KEY,
                type   ENUM('motoriste','avionneur','systemes')
                       NOT NULL,
                ville   VARCHAR(15) NOT NULL ,
                pays   VARCHAR(15) NOT NULL
 )
```

On note qu'il y a deux types de champs. **VARCHAR(long)** qui indique une variable de charactères de longueur variable avec un maximum de **long**. **INT** indique une variable de type entier. Le champ de type **ENUM** indique un choix parmi une liste pré-établie. Les paramètres servent à qualifier ou décrire le champ. Par exemple, le paramètre **NOT NULL** indique que ce champ doit être obligatoirement rempli. Le paramètre **NULL** indique que ce champ peut être vide, ce qui est différent de 0. **UNSIGNED** précise que la variable numérique doit être positive (ou zéro).

Une présentation plus complète des divers types de données se trouve à la Section 2.13.

De façon semblable pour la table des appareils (**avions**), nous avons :

```
CREATE TABLE avions
 (
              modele   VARCHAR(10) NOT NULL ,
              idAvion   INT UNSIGNED NOT NULL
                        AUTO_INCREMENT PRIMARY KEY,
    idConstructeur   INT NOT NULL REFERENCES constructeurs,
         nbMoteurs   INT UNSIGNED NOT NULL ,
          nbSieges   INT UNSIGNED NOT NULL ,
)
```

La démarche est illustrée avec les étapes suivantes :

1. Au départ, il n'y a pas de tables :

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

2. On crée la table :

```
mysql> CREATE TABLE avions
    -> (
    -> modele VARCHAR(10) NOT NULL,
    -> idAvion INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY ,
    -> idConstructeur INT(10) NOT NULL REFERENCES constructeurs,
    -> nbMoteurs INT UNSIGNED NOT NULL,
    -> nbSieges INT  UNSIGNED NOT NULL
    -> );
Query OK, 0 rows affected (0.00 sec)
```

3. On vérifie que la table a été effectivement créée

```
mysql> SHOW TABLES;
+-----------------+
| Tables_in_tixxx |
+-----------------+
| avions          |
+-----------------+
1 row in set (0.00 sec)
```

Dans le champ **id_avion**, on attribue un identifiant unique. Cette variable est de type **INT**, nombre entier, positif et obligatoire. Avec l'attribut **AUTO_INCREMENT**, on indique au serveur, lors de la création d'un nouvel enregistrement, de générer automatiquement une valeur distincte des autres dans ce champ. Finalement, l'attribut **PRIMARY KEY** indique un champ indexé pour une recherche rapide, et un identifiant unique qui évite les erreurs de doublons.

On répète de façon semblable pour la table des moteurs :

```
CREATE TABLE moteurs
 (
            modele   VARCHAR(15) NOT NULL ,
          idMoteur   INT UNSIGNED NOT NULL,
                     AUTO_INCREMENT PRIMARY KEY,
     idConstructeur  INT NOT NULL REFERENCES constructeurs,
          diametre    FLOAT,
          longueur    FLOAT,
             poids    FLOAT,
           poussee    FLOAT,
 )
```

La table de la flotte, **flotte**, identifie chacun des avions :

```
CREATE TABLE flotte
 (
     noSerieMoteur   VARCHAR(15) NOT NULL ,
      noSerieAvion   VARCHAR(10) NOT NULL ,
         idMoteur    INT NOT NULL REFERENCES moteurs ,
          idAvion    INT NOT NULL REFERENCES avions ,
       idSystemes    INT ,
          vitesse    FLOAT,
        autonomie    FLOAT,
      dateService    DATE NOT NULL ,
        heuresVol    FLOAT,
      dateRetrait    DATE NULL DEFAULT "0000-00-00",
     PRIMARY KEY     (noSerieAvion, noSerieMoteur)
 )
```

Les champs de type **DATE** donne une date selon la convention "année-mois-jour" avec AAAA-MM-JJ. La variable **date_service** est obligatoire, tandis que la variable **date_retrait** peut être vide car on ne connait pas au moment de la création de la BD, la date à laquelle l'avion sera retiré du service. Donc une date **NULL** indique un avion encore en service. On note que la clé primaire (obligatoire pour chaque table) est constituée par la concaténation de deux champs, **noSerieAvion** et **noSerieMoteur**.

Ces table peuvent être crées directement à la ligne de commande, ou bien, comme décrit à la Section 2.6, on peut à l'aide d'un éditeur générer un script pour la création de chacune de ces tables et les soumettre au serveur MySQL par :

```
$ mysql  ma_BD < creerConstructeurs.sql
$ mysql  ma_BD < creerAvions.sql
$ mysql  ma_BD < creerFlotte.sql
$ mysql  ma_BD < creerMoteurs.sql
```

ou bien à l'aide de la commande **SOURCE**, on soumet les fichiers précédents qui contiennent les instructions pour la création des différentes tables :

```
mysql> SOURCE creerAvions.sql;
Query OK, 0 rows affected (0.00 sec)

mysql> SOURCE creerConstructeurs.sql;
Query OK, 0 rows affected (0.00 sec)

mysql> SOURCE creerFlotte.sql;
Query OK, 0 rows affected (0.00 sec)

mysql> SOURCE creerMoteurs.sql;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW TABLES;
+-----------------+
| Tables_in_tixxx |
+-----------------+
| avions          |
| constructeurs   |
| flotte          |
| moteurs         |
+-----------------+
4 rows in set (0.00 sec)

mysql>
```

2.9 Gestion de la base de données

Pour vérifier la BD et son contenu, et examiner la structure des tables effectivement créées, on dispose de plusieurs commandes, à partir du client **mysql**,

mysql > SHOW DATABASES ;

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| H2006    |
| baron    |
| cae      |
| mysql    |
| test     |
| ti101    |
| ti102    |
| ti103    |
| ti104    |
| .....    |
| .....    |
| ti323    |
| ti324    |
| tixxx    |
| ts2h06   |
+----------+
78 rows in set (0.00 sec)

mysql>
```

mysql > SHOW TABLES ;

affiche la liste des tables appartenant à la base de données.

```
mysql> SHOW TABLES;
+-----------------+
| Tables_in_tixxx |
+-----------------+
| Etudiants       |
| moteurs         |
+-----------------+
2 rows in set (0.00 sec)
```

La commande **DESCRIBE**

```
DESCRIBE nom_table ;
```

affiche la structure du contenu tel que décrit lors de la création :

```
mysql> DESCRIBE Etudiants;
+-----------+------------------+------+-----+---------+----------------
| Field     | Type             | Null | Key | Default | Extra
+-----------+------------------+------+-----+---------+----------------
| matricule | int(10) unsigned |      | PRI | NULL    | auto_increment
| nom       | varchar(50)      | YES  |     | NULL    |
| prenom    | varchar(50)      | YES  |     | NULL    |
| sexe      | enum('F','M')    |      |     | F       |
+-----------+------------------+------+-----+---------+----------------
4 rows in set (0.00 sec)
```

Avec la commande **DROP TABLE**, on supprime le contenu et la structure d'une table :

```
DROP TABLE nom_table ;
```

Dans l'exemple suivant, on montre les tables présentes dans la BD, suivi de la commande **DROP TABLE moteurs ;**, et finalement, on vérifie que la table a été effectivement supprimée.

```
mysql> SHOW TABLES;
+-----------------+
| Tables_in_tixxx |
+-----------------+
| Etudiants       |
| moteurs         |
+-----------------+
2 rows in set (0.00 sec)

mysql> DROP TABLE moteurs;
Query OK, 0 rows affected (0.02 sec)

mysql> SHOW TABLES;
+-----------------+
```

```
| Tables_in_tixxx |
+-----------------+
| Etudiants       |
+-----------------+
1 row in set (0.00 sec)
```

Ces commandes permettent de consulter la BD pour déterminer son contenu et la structure interne des tables. La commande **DESCRIBE** est particulièrement utile lorsque l'on veut modifier ou ajouter des enregistrements et que l'on ne se souvient plus des différents champs. Ces fonctions sont disponibles à l'intérieur du client mysql ou à partir du shell Unix :

Shell Unix	Client mysql
$ **mysqlshow**	**SHOW DATABASES**
$ **mysqlshow nom_BD**	**SHOW TABLES**
$ **mysqlshow nom_BD nom_table**	**DESCRIBE nom_table**

TAB. 2.6 – Commandes de consultation d'une BD

2.10 Les enregistrements

Après avoir créé une BD et déclaré des tables avec des champs correspondant à des type de données précis, nous avons une structure qui organise les données qu'il faut remplir. On peut introduire des données dans une BD manuellement avec la commande **INSERT**, ou en lecture à partir d'un fichier informatique avec l'instruction **LOAD DATA** ou avec un script *.sql qui contient des énoncés **INSERT**.

À l'inverse, pour récupérer ou consulter des enregistrements existants, on utilise l'instruction **SELECT** (Voir Section 2.11).

La commande **INSERT** permet de remplir une table en y insérant des enregistrements.

```
INSERT   INTO nom_table
         VALUES(champ_1, champ_2, champ_3,... )
```

La liste **VALUES** contient une valeur pour chacun des champs de la table, dans l'ordre dans lequel ils ont été donnés lors de sa création, et avec le type et le domaine appropriés. On illustre l'ajout de l'acquisition d'un nouvel appareil

dans l'exemple de la flotte (2.1) au Tableau 2.5 :

```
mysql> INSERT INTO flotte
VALUES("2367H","dg789-9",1,2,NULL,800,6000,1999,1200,NULL);

Query OK, 1 row affected (0.01 sec)
```

Ce qui donne le résultat suivant qui montre l'ajout de cet enregistrement à la fin de la liste :

```
mysql> select * FROM flotte;
+---------------+--------------+----------+---------+------------
| noSerieMoteur | noSerieAvion | idMoteur | idAvion | idSystemes
+---------------+--------------+----------+---------+------------
| 1358D         | SL120-8      |        1 |       2 |       NULL
| 2945F         | pw340-A      |        2 |       5 |       NULL
| 2367H         | dg789-9      |        1 |       2 |       NULL
+---------------+--------------+----------+---------+------------
```

```
--+---------+-----------+-------------+-----------+-------------+
  | vitesse | autonomie | dateService | heuresVol | dateRetrait |
--+---------+-----------+-------------+-----------+-------------+
  |         |      4500 | 0000-00-00  |      2300 | 0000-00-00  |
  |         |      5600 | 0000-00-00  |      1500 | NULL        |
  |         |      6000 | 0000-00-00  |      1200 | NULL        |
--+---------+-----------+-------------+-----------+-------------+

3 rows in set (0.00 sec)

mysql>
```

Dans une même commande **INSERT**, on peut entrer plusieurs enregistrements :

```
mysql > INSERT INTO flotte VALUES
        ("3471H", 4, 2006-03-13,0,NULL ),
        ("5471R", 3, 2006-03-13,0,NULL ),
        ("6371T", 3, 2006-03-13,0,NULL );
```

Lorsque les données sont volumineuses, il est utile de le faire dans un fichier *.sql, et ensuite de le soumettre comme un script à partir du shell UNIX :

```
$ mysql  flotte_BD < insert_flotte.sql ;
```

ou bien avec la commande **SOURCE** :

```
$ mysql  SOURCE insert_flotte.sql ;
```

On peut modifier ou éliminer des enregistrements avec les commandes **DELETE** et **UPDATE** :

```
$ mysql  DELETE FROM table WHERE enregistrements
```

Sans la clause **WHERE**, la commande delete supprime TOUS les enregistrements ! Donc, à utiliser avec soin. Une précaution est de tester son action au préalable avec la commande **SELECT** qui affichera le résultat de la recherche. Si cela correspond à l'action souhaitée, alors on peut émettre la commande **DELETE** avec la même option **WHERE**.

La syntaxe pour la commande **UPDATE** est la suivante :

```
$ mysql  UPDATE table SET champs WHERE enregistrements
```

Ici encore, sans la clause **WHERE**, la commande **UPDATE** remplacera TOUS les champs avec les valeurs données dans **SET**.

```
UPDATE lesAppareils
               SET statut="enEMPRUNT", responsable="identifiant"
               WHERE noSerie="123A";
```

Les champs `statut` et `responsable` de la table `lesAppareils` dont les enregistrements contienent un champ `noSerie` égal à `123A`, seront remplacés par les valeurs `enEMPRUNT` et `identifiant`.

2.11 Interrogation de la BD

L'extraction des données contenues dans une BD se fait à l'aide de la commande **SELECT**. Dans sa forme de base, on précise les données voulues, la table où les chercher et les conditions que ces dernières doivent vérifier :

```
SELECT   données
FROM     nom_table
WHERE    conditions
```

Les **données** sont les informations contenues dans les champs de la table.

2.11.1 Recherche dans une table

Pour extraire toutes les données contenues dans une table, on utilise une forme simplifiée de cette commande :

```
SELECT *   FROM nom_table
```

qui affichera tout le contenu de la table sous le format utilisé lors de la création de la table avec la commande **CREATE TABLE** (Voir Section 2.8). Le paramètre ***** indique tous les champs de tous les enregistrements.

```
mysql> SELECT * FROM constructeurs;
+--------------------+-----+-----------+-------------+-----------+
| nomCompagnie       | idC | type      | ville       | pays      |
+--------------------+-----+-----------+-------------+-----------+
| Boeing Co.         |   1 | avionneur | Chicago     | USA       |
| Bombardier Aero.   |   2 | avionneur | Montreal    | Canada    |
| Airbus Industries  |   3 | avionneur | Toulouse    | France    |
| General Electric   |   4 | motoriste | Schenecdaty | USA       |
| Pratt & Whitney    |   5 | motoriste | Hartford    | USA       |
| Rolls Royce        |   6 | motoriste | Derby       | Angleterre|
+--------------------+-----+-----------+-------------+-----------+

6 rows in set (0.01 sec)
```

On sélectionne le contenu d'une ou de plusieurs colonnes (champs) en les explicitant, (séparés par des virgules) :

```
SELECT   champs_1, champs_2
FROM     nom_table
```

Cette commande affichera tous les champs ***champs_1*** et ***champs_2*** de tous les enregistrements de la table ***nom_table***.

Un critère permet de sélectionner un enregistrement en particulier dans la table. La condition est appliquée par l'option **WHERE**. Par exemple on peut sé-

lectionner, dans une colonne (champ), *champs_N* , les enregistrements (rangées) pour lesquels le champs *champs_M* vérifie la condition *champs_M = 'Valeur'* :

```
SELECT   champs_N
FROM     nom_table
WHERE    champs_M = 'Valeur'
```

Les critères de sélection s'expriment par des expressions qui sont composées d'opérations arithmétiques ou logiques, et d'opérateurs relationnels, résumés au Tableau 2.7.

Opérations arithmétiques		Opérations logiques		Opérateurs relationnels	
+	Addition	**AND**	**ET** logique	=	égal
-	Soustraction	**OR**	**ou** logique	**!=**	pas égal
*	Multiplication	**NOT**	**négation** logique	<	plus petit
/	Division			>	plus grand

TAB. 2.7 – Opérations de sélection

```
mysql> SELECT nomCompagnie
    -> FROM constructeurs
    -> WHERE pays="Canada";
+--------------------------+
| nomCompagnie             |
+--------------------------+
| Bombardier Aeronautique  |
+--------------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT modele
    -> FROM avions
    -> WHERE nbSieges > 200;
+----------+
| modele   |
+----------+
| B747-400 |
| B757-200 |
| A340-600 |
| A340-500 |
| A330-300 |
| A330-200 |
| A310-300 |
+----------+
7 rows in set (0.00 sec)
```

Souvent lors d'une requête, il est pratique d'utiliser la commande **DESCRIBE** pour se rappeller la structure de la table, ainsi que le libellé des champs :

```
mysql> DESCRIBE avions;
+----------------+------------------+------+-----+---------+----------------+
| Field          | Type             | Null | Key | Default | Extra          |
+----------------+------------------+------+-----+---------+----------------+
| modele         | varchar(10)      |      |     |         |                |
| idAvion        | int(10) unsigned |      | PRI | NULL    | auto_increment |
| idConstructeur | int(10)          |      |     | 0       |                |
| nbMoteurs      | int(10) unsigned |      |     | 0       |                |
| nbSieges       | int(10) unsigned |      |     | 0       |                |
+----------------+------------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

Les champs contenant des valeurs **NULL** ne peuvent se comparer avec les opérateurs du Tableau 2.7, mais nécessitent un traitement particulier. En effet, comme la valeur **NULL** indique aucune valeur, il n'est pas possible de comparer deux champs qui n'ont pas de valeur ! Cependant on identifier un champ de type **NULL** à l'aide d'un opérateur spécial **IS NULL** ou **IS NOT NULL**

2.11.2 Présentation des résultats

Les résultats d'une requête sont généralement affichés dans l'ordre dans lequel les enregistrements ont été insérés dans la table. Cependant lors de différentes manipulations, certains enregistrements seront enlevés et d'autres ajoutés. Pour rationnaliser l'utilisation de l'espace mémoire, MySQL tentera d'insérer les nouveaux enregistrements dans les espaces laissés vides par la suppression d'enregistrements. De sorte que par défaut, il n'y a pas de garantie quant à l'ordre des enregistrements dans une table.

Pour un affichage correct, il faut les trier avec l'option **ORDER BY**, qui prend l'argument **ASC** ou **DESC**, pour un ordre ascendant ou descendant, respectivement.

```
SELECT champs1, champs2
FROM maTable
ORDER BY champs1 ASC
```

La fonction **COUNT()** permet de compter des d'enregistrements. Dans sa version la plus simple, où le paramètre ***** signifie "TOUS", **COUNT(*)** retourne le nombre total d'enregistrements dans une table :

```
mysql> SELECT COUNT(*) FROM moteurs;
+----------+
| count(*) |
+----------+
|       11 |
+----------+
1 row in set (0.00 sec)
```

On peut relier ce décompte à une condition, par exemple à la valeur d'un champ, et ceci dans plusieurs tables grâce aux relations :

```
 mysql> SELECT COUNT(*)
     -> FROM moteurs,constructeurs
     -> WHERE constructeurs.pays="USA"
     -> AND constructeurs.idConstructeur=moteurs.idConstructeur;
+----------+
| count(*) |
+----------+
|        3 |
+----------+
1 row in set (0.00 sec)
```

Il est possible de produire divers résumés par regroupement en fonction de certaines catégories avec l'option **GROUP BY** :

```
mysql> SELECT type  FROM constructeurs GROUP BY type;
+-----------+
| type      |
+-----------+
| motoriste |
| avionneur |
+-----------+
2 rows in set (0.01 sec)
```

On peut faire un décompte par catégorie en combinant l'option **GROUP BY** avec **COUNT()** :

```
mysql> SELECT type, COUNT(*) FROM constructeurs GROUP BY type;
+-----------+----------+
| type      | COUNT(*) |
+-----------+----------+
| motoriste |        3 |
| avionneur |        3 |
+-----------+----------+
2 rows in set (0.00 sec)
```

Finalement, ce type de résumé peut se faire au travers de plusieurs tables ; par exemple on compte, dans la table ***avions***, le nombre d'appareils produits par pays :

```
mysql> SELECT nomCompagnie,pays, COUNT(*)
    -> FROM avions, constructeurs
    -> WHERE avions.idConstructeur=constructeurs.idConstructeur
    -> GROUP BY pays;
+-------------------+--------+----------+
| nomCompagnie      | pays   | count(*) |
+-------------------+--------+----------+
| Airbus Industries | France |        5 |
| Boeing Co.        | USA    |        3 |
+-------------------+--------+----------+
2 rows in set (0.00 sec)
```

Une analyse de l'état de la flotte peut être réalisée à partir de données telles que le nombres de sièges et le nombre d'heures accumulées.

```
mysql> SELECT modele, nbSieges, heuresVol
    -> FROM flotte, avions
    -> WHERE avions.idAvion=flotte.idAvion;
+----------+----------+-----------+
| modele   | nbSieges | heuresVol |
+----------+----------+-----------+
| B757-200 |      243 |      2300 |
| A340-500 |      313 |      1500 |
| B757-200 |      243 |      1200 |
+----------+----------+-----------+
3 rows in set (0.00 sec)
```

Pour plus de réalisme on ne devrait compter que les appareils encore en service (ceux possédant une valeur NULL) :

```
mysql> SELECT modele, nbSieges, heuresVol
    -> FROM flotte, avions
    -> WHERE avions.idAvion=flotte.idAvion
    -> AND dateRetrait IS NULL;
+----------+----------+-----------+
| modele   | nbSieges | heuresVol |
+----------+----------+-----------+
| A340-500 |      313 |      1500 |
| B757-200 |      243 |      1200 |
+----------+----------+-----------+
2 rows in set (0.00 sec)
```

Avec les fonctions de calcul plusieurs opérations sur les valeurs de colonnes

(champs) sont possible, tel que calculer le minimun (maximun), la moyenne, la somme etc.... Par exemple, on pourrait calculer la capacité totale et l'"age" de la flotte par la somme des sièges et de la somme des heures de vol :

```
mysql> SELECT SUM(nbSieges) AS Sieges, SUM(heuresVol) AS Heures
    -> FROM flotte, avions
    -> WHERE avions.idAvion=flotte.idAvion
    -> AND dateRetrait IS NULL;
+--------+--------+
| Sieges | Heures |
+--------+--------+
|    556 |   2700 |
+--------+--------+
1 row in set (0.00 sec)
```

2.11.3 Recherche dans plusieurs tables

Afin d'exploiter pleinement la structure relationnelle des tables, il faut pouvoir rechercher des informations dans plusieurs tables. Le mécanisme qui permet cette opération s'appelle une *jointure* et s'interprête comme la jonction des valeurs communes. On l'exprime par une condition sur la relation entre les clés primaire et étrangère des tables mises en jeu. Comme souvent ces dernières portent le même nom, on les différencie par leur nom complet :

nomTable.nomChamp

La liste des avions de la flotte, on affiche le modèle, le constructeur, son pays et le nombre de sièges :

```
mysql> SELECT modele,nomCompagnie,pays,nbSieges
    -> FROM avions, constructeurs
    -> WHERE avions.idConstructeur=constructeurs.idConstructeur;
+----------+-------------------+--------+----------+
| modele   | nomCompagnie      | pays   | nbSieges |
+----------+-------------------+--------+----------+
| B747-400 | Boeing Co.        | USA    |      420 |
| B757-200 | Boeing Co.        | USA    |      243 |
| B727     | Boeing Co.        | USA    |      160 |
| A340-600 | Airbus Industries | France |      380 |
| A340-500 | Airbus Industries | France |      313 |
| A330-300 | Airbus Industries | France |      362 |
| A330-200 | Airbus Industries | France |      363 |
| A310-300 | Airbus Industries | France |      259 |
+----------+-------------------+--------+----------+
8 rows in set (0.00 sec)
```

On peut enrichir la requête, avec plusieurs conditions :

```
mysql> SELECT modele, nomCompagnie
    -> FROM avions,constructeurs
    -> WHERE nbSieges > 300
    -> AND avions.idConstructeur=constructeurs.idConstructeur;

+----------+-------------------+
| modele   | nomCompagnie      |
+----------+-------------------+
| B747-400 | Boeing Co.        |
| A340-600 | Airbus Industries |
| A340-500 | Airbus Industries |
| A330-300 | Airbus Industries |
| A330-200 | Airbus Industries |
+----------+-------------------+
5 rows in set (0.00 sec)
```

2.12 Édition des tables

Au cours du développement d'un projet de base de données, et de son utilisation, il est nécessaire de modifier les éléments de la base de données. Ces

modifications peuvent être du niveau orthographique ou bien structurel. On peut vouloir corriger l'orthographe des variables, en ajouter ou en retrancher ; modifier les charactéristiques des champs etc.. Par exemple, un champ a été défini avec un type CHAR(100) et il s'est avéré en cours de route que certaines valeurs dépassent cette taille qui devra donc être modifiée. À l'inverse, si à l'usage aucune valeur ne dépasse la taille originalement précisée, on peut la diminuer pour économiser de l'espace mémoire.

```
ALTER TABLE nom_table MODIFICATION ;
```

Avec cette structure de commande, on identifie la table et la colonne (champ) où la modification aura lieu, suivis de la modification et des paramètres. La commande **ALTER TABLE** offre plusieurs possibilités :

Renommer une table : On spécifie l'ancien et le nouveau nom de la table :

```
ALTER TABLE ancien_nom RENAME AS nouveau_nom ;
```

modifier le type de champs :

```
ALTER TABLE nom_table MODIFY variable FLOAT ;
```

L'option **MODIFY** modifie le type du champ "variable" en **FLOAT**

Renommer un champ : Le nom d'une variable ou d'un champ peut être changé par :

```
ALTER TABLE nom_table CHANGE titre_1 titre_2 ;
```

2.13 Types de données

Tous les éléments des tables dans une BD sont des données de type prédéfinis, et ceux-ci déterminent les manipulations et opérations possibles. Le choix d'un type de donnée pour un champ dépend de la nature de la quantité représentée, et ceci constitue une étape importante dans l'élaboration de la table. Ces types sont spécifiés lors de la création des tables avec la syntaxe suivante :

```
CREATE TABLE nom_de_table
 (
    nomChamp   typeChamp   [attributsChamps]   [attributsGeneraux]
 )
```

Le champ **nomChamp** explicite le nom qui est constitué de caractères alphanumériques, mais ne peut pas comprendre uniquement des chiffres. Le champ **typeChamp** représente le type de données qui sera inscrit dans la table dans ce champ. Parmi les possibilités : des nombres, des caractères, des dates etc...Selon le type on pourra préciser avec des attributs qui qualifient le type et des attributs généraux qui s'appliquent à tous les types. Par exemple, l'attribut général **NULL** indique une valeur non saisie. Ceci signifie qu'il n'y a aucune valeur et non pas que la valeur est "zéro" ! L'attribut **NOT NULL** signifie qu'il doit obligatoirement avoir une valeur dans le champ en question, c'est -à-dire qu'il ne peut être laissé vide.

Les nombres

MySql permet 5 types d'entiers et 3 types de "réels" décrits au Tableau 2.8 avec les plages de valeurs correspondantes. Cette variété est nécessaire pour ajuster la longueur des champs en fonction de la taille du nombre à représenter et ainsi donner une utilisation optimale de la mémoire. Cependant, si le nombre dépasse la taille permise par le type déclaré alors, il sera tronqué.

Type	*Description*	SIGNED	UNSIGNED
TINYINT	entier	-2^7 à 2^7-1	0 à 2^8-1
SMALLINT	entier	-2^{15} à $2^{15}-1$	0 à $2^{15}-1$
MEDIUMINT	entier	-2^{23} à $2^{23}-1$	0 à $2^{24}-1$
INT	entier	-2^{31} à $2^{31}-1$	0 à $2^{32}-1$
BIGINT	entier	-2^{63} à $2^{63}-1$	0 à $2^{64}-1$
FLOAT	"réel" simple	$\pm 1.175494351E-38$	
	précision	$\pm 3.402823466E+38$	
DOUBLE	"réel" double	$\pm 2.2250738585072014E-308$	
	précision	$\pm 1.7976931348623157E+308$	

TAB. 2.8 – Les différents types de nombres

Parmi les attributs possibles au type *nombre*, deux sont particulièrement utiles :
– ***AUTO_INCREMENT*** permet de générer automatiquement des suites de valeurs uniques. En spécifiant cet attribut, les valeurs du champ sont au-

tomatiquement générées à partir de la valeur 1 et sont augmentées par 1 à chaque nouveau enregistrement. Normalement, on déclare le champ comme **NOT NULL** et comme une clé primaire **PRIMARY KEY**.
- **UNSIGNED** permet de spécifier des valeurs positives ;
- **NOT NULL**
- **DEFAULT** permet de spécifier une valeur qui sera prise comme valeur par defaut ; normalement, pour un champ numérique la valeur par défaut est égal à **NULL**. Pour un autre choix, il faut le préciser avec l'attribut **DEFAULT** :

```
CREATE TABLE t
 (
    i   INT   DEFAULT   1
 )
```

Chaînes de caractères

Ce type de champs sert à représenter des caractères et n'importe quelle données binaires (fichiers, images, son etc).

Type	*Description*	*Taille maximale*
CHAR(L)	chaîne de caractères de longueur fixe	L octets
VARCHAR(L)	chaîne de caractères de longueur variable	L octets
ENUM	énumération de membres	65 53 membres
SET	Ensemble	64 membres

TAB. 2.9 – Les différents types de Chaînes de caractères

- **CHAR** et **VARCHAR** désignent deux types de chaînes. Dans le premier cas, chaque champ occupe un espace mémoire de longueur fixe et si la valeur est inférieure à la taille de champ spécifiée, elle sera complétée par des espaces vides. Avec **VARCHAR**, les champs occupent autant d'espaces que nécessaire. On utilisera la première forme si les valeurs ne varient pas beaucoup entre elles. Par contre, si il y a une grande variabilité d'une valeur à l'autre la seconde forme sera une approche plus efficace sur le plan de la gestion de la mémoire.
- **ENUM** et **SET** spécifient des valeurs qui sont choisies à partir d'une liste de

valeurs pré-établies. Avec **ENUM**, les valeurs sont mutuellement exclusives, et on sélectionne une seule valeur parmi l'ensemble. **SET** permet de spécifier un sous-ensemble parmi les possibilités. On s'en servira pour définir des choix multiples.

```
CREATE TABLE t
 (
    couleur      ENUM('rouge','vert','jaune')
    taille       ENUM('grand','moyen','petit')
    garnitures   SET('champignons','fromage','anchois','peperoni')
 )
```

Date

DATE	Une date dans le format aaaa-mm-jj
TIME	L'heure dans le format hh :mm :ss
TIMESTAMP	Une durée dans le format aaaammjjhhmmss

