.1 Intérêt de SQL.

Tous les systèmes de gestion de données utilisent SQL pour l'accès aux données ou pour communiquer avec un serveur de données. SQL (Standard Query Language) est né à la suite des travaux mathématiques de Codd, travaux qui ont fondé les bases de données relationnelles. SQL, défini d'abord chez IBM, a subi trois tentatives de normalisation en 86, 89 et 92 (SQL 2 ou SQL 92). Nous présentons trois raisons fondamentales qui justifient l'utilisation de SQL.

- D'une part, la structuration et la manipulation des données sont devenues très complexes. Pour une application de taille moyenne, la base de données contient fréquemment plus de trente tables fortement interconnectées. Il est donc hors de question de manipuler les données de façon algorithmique traditionnelle. Une requête SQL dans un langage logique simple remplace donc bien avantageusement plusieurs dizaines de lignes d'un langage de programmation tel C ou Cobol.

- D'autre part, l'architecture client-serveur est omniprésente. Tandis que la station client exécute le code de l'application en gérant, en particulier, l'interface graphique, le serveur optimise la manipulation et le contrôle des données. De plus, les applications doivent être portables et gérer des données virtuelles, c'est-à-dire émanant de n'importe quel serveur. Développer une application dans un environnement hétérogène n'est possible que parce que la communication entre l'applicatif client et le serveur est réalisée par des primitives SQL normalisées.
- Enfin, les applications à développer (même sur un PC) sont devenues de plus en plus complexes. Le profil du programmeur a fortement changé. Il doit maintenant traiter des données de plus en plus volumineuses, intégrer les techniques de manipulation des interfaces, maîtriser la logique événementielle et la programmation orientée objet, tout celà dans un contexte d'architecture client-serveur où se cotoient les systèmes d'exploitation et les protocoles de réseaux hétérogènes. L'accès et la manipulation des données ne sont que l'un des aspects de la conception et de la réalisation de programmes. On cherche donc à acquérir un environnement de développement performant qui prend en charge un grand nombre de tâches annexes. Des outils de développement sont apparus pour permettre au développeur de se concentrer sur l'application proprement dite : générateurs d'écrans, de rapports, de requêtes, d'aide à la conception de programme, de connexion à des bases de données distantes via des réseaux. Dans tous ces outils, la simplicité et la standardisation de SQL font que SQL est utilisé chaque fois qu'une définition, une manipulation, ou un contrôle de données est nécessaire. SQL est donc un élément central entre les divers composants d'un environnement de développement dans une architecture client-serveur.

.2 SQL dans l'architecture en couches des SGBD.

Dans la phase d'analyse de systèmes d'information, on considère différents niveaux d'abstraction du système d'information : le niveau conceptuel, le niveau logique ou organisationnel et enfin, le niveau physique ou opérationnel.

Nous allons considérer ici différents niveaux de perception d'une base de données relationnelle. Un SGBD est fréquemment décrit par une structure en couches correspondant à des perceptions différentes des données, associées à des tâches différentes pour différents

acteurs.

Pour plus de simplicité, nous distinguerons trois types d'acteurs : les administateurs de la base de donnée, les développeurs et les utilisateurs. . Au niveau externe, proche de l'utilisateur, la perception est totalement indépendante du matériel et des techniques mises en oeuvre, tandis qu'au niveau le plus intérieur, se trouvent les détails de l'organisation sur disque et en mémoire.

Le *schéma logique* est l'ensemble de toutes les données pertinentes, toutes applications confondues. Il est rendu conforme à un modèle de représentation des données, et est totalement indépendant de la technologie utilisée. Nous choisissons le modèle relationnel. Ce niveau a un inconvénient : toutes les données sont accessibles à tout le monde. Cet ensemble vaste de données est trop touffu. Il est préférable de montrer à l'utilisateur (et au programmeur) une vue plus simple des données. On constitue ainsi des *schémas externes*. Par exemple, le gestionnaire du stock n'est concerné que par les données décrivant les articles en stock. S'il ne manipule que des bordereaux d'entrée, des bordereaux de sortie, et des fiches d'état du stock, ceux-ci constituent son schéma de perception externe. Le *schéma interne* fournit une perception plus technique des données, et enfin le *schéma physique* est dépendant du matériel et du logiciel de base.

Niveau externe.
: Au niveau externe, les utilisateurs et développeurs d'application ont une perception limitée de la base de données. On parle de *vue*. Une vue peut être considérée comme une restriction du schéma logique à un type d'utilisateur. Ce niveau concerne les utilisateurs et les développeurs.

Niveau logique.
: Traduction dans le modèle relationnel du schéma conceptuel. On précise à ce niveau les tables, les relations entre tables, les contraintes d'intégrité, les vues et les droits par groupe d'utilisateurs. Ce niveau concerne l'administrateur et les développeurs.

Niveau interne.
: On définit les index et tous les éléments informatiques susceptibles d'optimiser les ressources et les accès aux données. Ce niveau concerne l'administrateur.

Niveau physique.
: On y précise tout ce qui dépend du matériel et du système d'exploitation. Ce niveau concerne l'administrateur.

Cette découpe en niveaux présente les avantages suivants :

- Les applications développées sont indépendantes du niveau interne. Tout changement de stratégie d'accès, ou d'organisation des données entraîne une modification au niveau interne, mais le schéma logique reste inchangé. Par exemple, une requête SQL précise le QUOI sans se préoccuper du COMMENT.
- La distinction externe/logique assure (en partie) l'indépendance entre les applications et le niveau logique. Par exemple on peut enrichir le schéma logique sans modifier les applications existantes pour toutes les vues non concernées par les modifications apportées au schéma logique.
- La distinction logique/interne permet de modifier les optimisations d'accès aux données. Par exemple, si une application a des performances insuffisantes, il est possible d'optimiser les accès (en introduisant de nouveaux index, par exemple) et d'augmenter les performances sans modifier l'application.

- La distinction interne/physique permet une meilleure portabilité car seule la partie physique est dépendante du matériel et du système d'exploitation.

.3 Principes de base d'une Base de Données relationnelle

Le terme *relationnel* provient de la définition mathématique d'algèbre relationnelle (Codd 70). Une relation est un ensemble de tuples[4] de données, on peut alors définir des opérations algébriques sur les relations . Nous parlerons dans la suite de table, et non pas de relation. Une *table* est un ensemble de tuples de données distincts deux à deux. Une table est constituée d'une *clé primaire* et de plusieurs *attributs* (ou colonnes) qui dépendent de cette clé. La clé primaire d'une table est un attribut ou un groupe d'attributs de la table qui détermine tous les autres de façon unique. Une table possède toujours une et une seule clé primaire. Par contre, une table peut présenter plusieurs clés candidates qui pourraient jouer ce rôle. Le *domaine d'un attribut* est l'ensemble des valeurs que peut prendre cet attribut. Le domaine est constitué d'un type, d'une longueur et de contraintes qui réduisent l'ensemble des valeurs permises. Une clé étrangère dans une table est une clé primaire ou candidate dans une autre table. Les *contraintes d'intégrité* font partie du schéma logique. Parmi celles-ci, on distingue

- les *contraintes de domaine* qui restreignent l'ensemble des valeurs que peut prendre un attribut dans une table,
- les *contraintes d'intégrité d'entité* qui précisent qu'une table doit toujours avoir une clé primaire et
- les *contraintes d'intégrité référentielles* qui précisent les conditions dans lesquelles peuvent être ajoutés ou supprimés des enregistrements lorsqu'il existe des associations entre tables par l'intermédiaire de clés étrangères.

Exemple 1

***CLIENTS (cltnum**, cltnom, cltpnom, cltloc, cltca, clttype)*
***COMMANDES (cmdnum**,* cmdclt*, cmddate, cmdvnd)*
***LIGCOMMANDES (ldccmd, ldcart**, ldcqte)*
***ARTICLES (artnum**, artnom, artpv, artcoul)*

Les identifiants sont en gras, les clés étrangères en italique. Une contrainte d'intégrité référentielle est, par exemple, l'obligation de la présence d'un client pour une comande. C'est-à-dire encore qu'à un enregistrement dans la table COMMANDES doit correspondre un enregistrement de la table CLIENTS tel que

COMMANDES.cmdclt=CLIENTS.cltnum.

.4 Architecture client-serveur et communication par SQL

Nous allons d'abord revoir les différents types d'application possibles pour la gestion de données distantes en commençant par les trois formes les plus simples.

Monoposte.

La base de données se trouve sur un poste et n'est pas accessible en réseau. Il faut, dans ce cas, penser à la notion de sécurité si plusieurs utilisateurs peuvent interroger la base (suppression accidentelle d'enregistrements).

Multiposte, basée sur des terminaux

, liés à un site central. C'est l'informatique multiposte traditionnelle. La gestion des données est centralisée. Les applications ont été écrites par le service informatique et seules ces applications peuvent interroger le serveur.

Multiposte, basée sur un serveur de fichiers.

C'est la première forme (la plus simple) d'architecture client-serveur. Si l'applicatif sur un PC souhaite visualiser la liste des clients habitant Paris, tous les enregistrements du fichier CLIENT transitent sur le réseau, entre le serveur et le client, la sélection (des clients habitant Paris) est faite sur le PC. Le trafic sur le réseau est énorme et les performances se dégradent lorsque le nombre de clients augmente. Les serveurs de fichiers restent très utilisés comme serveurs d'images, de documents, d'archives.

De nouveaux besoins sont apparus :

- diminuer le trafic sur le réseau pour augmenter le nombre de postes sans nuire au fonctionnement,
- traiter des volumes de données de plus en plus grand,
- accéder de façon transparente à des données situées sur des serveurs différents,
- accéder aux données de façon ensembliste, même si les données sont distantes, afin de diminuer le travail du programmeur,
- adopter des interfaces graphiques de type Windows pour les applicatifs clients.

Bases de données en client-serveur

Dans une architecture client-serveur, un applicatif est constitué de trois parties : l'interface utilisateur, la logique des traitements et la gestion des données. Le client n'exécute que l'interface utilisateur et la logique des traitements, laissant au serveur de bases de données la gestion complète des manipulations de données.

- Le serveur de bases de données fournit des services aux processus clients. Les tâches qu'il doit prendre en compte sont : la gestion d'une mémoire cache, l'exécution de requêtes exprimées en SQL, exécuter des requêtes mémorisées, la gestion des transactions, la sécurité des données.

- Le client doit ouvrir une connexion pour pouvoir profiter des services du serveur. Il peut ouvrir plusieurs connexions simultanées sur plusieurs serveurs. Le client peut soumettre plusieurs requêtes simultanément au serveur et traiter les résultats de façon asynchrone.

- Communication entre le client et le serveur. Puisque l'application doit pouvoir se connecter à divers serveurs de façon transparente, le langage de communication SQL doit être compatible avec la syntaxe SQL de chaque serveur pressenti. Or, malgré les normes, les dialectes SQL sont nombreux et parfois source d'incompatibilité. La seule façon de permettre une communication plus large est d'adopter un langage SQL standardisé de communication. Une couche fonctionnelle du client traduit les requêtes du dialecte SQL client en SQL normalisé. La requête transformée est envoyée au serveur. Celui-ci traduit la requête dans le dialecte SQL-serveur et l'exécute. Le résultat de la requête suit le chemin inverse. Le langage de communication normalisé le plus fréquent est l'**ODBC** (Open DataBase Connectivity) de Microsoft. Signalons également **IDAPI** (Integrated Database Application Programming Interface) de Borland. De plus, ces programmes permettent d'interroger des bases de données autres que relationnelles.

Les serveurs de transactions

Une transaction correspond à une procédure SQL, i.e. un groupe d'instructions SQL. Il y a un seul échange requête/réponse pour la transaction. Le succès ou l'échec concerne l'ensemble des instructions SQL. Ces serveurs sont essentiellement utilisés pour l'informatique de production (ou opérationnelle) pour laquelle la rapidité des temps de réponse est importante sinon essentielle.

.5 Structure générale du langage SQL

Les instructions essentielles SQL se répartissent en trois familles fonctionnellement distinctes et trois formes d'utilisation :

Selon la norme SQL 92, le *SQL interactif* permet d'exécuter une requête et d'en obtenir immédiatement une réponse. Le *SQL intégré* (ou module SQL) permet d'utiliser SQL dans un langage de troisième génération (C, Cobol, ...), les instructions permettant essentiellement de gérer les curseurs. Le *SQL dynamique* est une extension du SQL intégré qui permet d'exécuter des requêtes SQL non connues au moment de la compilation. Hors norme, SQL est utilisable sous la forme de librairies de fonctions API (exemple : ODBC). Nous nous limitons au SQL interactif.

Dans le SQL interactif, le **LDD** (Langage de Définition de données) permet la description de la structure de la base (tables, vues, index, attributs, ...). Le dictionnaire contient à tout moment le descriptif complet de la structure de données. Le **LMD** (Langage de Manipulation de Données) permet la manipulation des tables et des vues. Le **LCD** (Langage de Contrôle des Données) contient les primitives de gestion des transactions et des privilèges d'accès aux données. Le tableau ci-dessous vous donne les principales primitives SQL et leur classification. Nous nous intéresserons essentiellement au LMD et à la commande SELECT. Nous étudierons également quelques problèmes concernant les privilèges d'accès et les transactions.

SQL interactif		
LDD	LMD	LCD
CREATE	SELECT	GRANT
DROP	INSERT	REVOKE
ALTER	DELETE	CONNECT
	UPDATE	COMMIT
		ROLLBACK
		SET

.6 SQL : un langage algébrique

Pour bien comprendre le langage SQL, nous allons brièvement exposer les principes sur lesquels repose ce langage (algèbre relationnelle).

Une table (relation) est un ensemble de tuples. On peut donc appliquer à une table les opérateurs algébriques usuels. Le résultat d'une opération ou requête est une nouvelle table qui est exploitable à son tour dans une nouvelle opération. Tous les opérateurs peuvent être dérivés de cinq primitives de base : la PROJECTION, la SELECTION, l'UNION, la

DIFFERENCE et le PRODUIT. L'opérateur de JOINTURE qui peut être déduit des cinq primitives de base est cependant fondamental.

La PROJECTION

permet de ne conserver que les attributs intéressants d'une table (sélection verticale). De plus, la projection élimine les répétitions de tuples résultant de cette sélection.

Exemple 2 *CLIENTS2 = **PROJECT** CLIENTS **OVER** (cltnom, cltloc)*

La SELECTION

permet de ne conserver que les tuples qui respectent une condition définie sur les valeurs des attributs (sélection horizontale).

Exemple 3 *BONSCLIENTS = **SELECT** CLIENTS **WHERE** cltca>10000*

L'UNION

réalise l'union de plusieurs tables.

Exemple 4 *CLIENTS3 = **SELECT** CLIENTS **WHERE** cltloc = 'PARIS' **UNION** **SELECT** BONSCLIENTS **WHERE** cltloc='BRUXELLES'*

La DIFFERENCE

consiste à prendre les tuples appartenant à une table mais pas à une autre.

Exemple 5 *CLIENTS4 = **SELECT** CLIENTS **WHERE** cltloc = 'BRUXELLES' **EXCEPT** BONSCLIENTS*

Le PRODUIT

réalise la juxtaposition de tous les tuples de la première table avec chaque tuple de la seconde. Celà signifie que, si les deux tables ont respectivement *M* et *N* tuples, la table résultante aura *M*× *N* tuples. Cette opération présente peu d'intérêt mais combinée avec une sélection, on obtient une opération fondamentale : la JOINTURE.

La JOINTURE

n'est possible que sur deux tables possédant un attribut de domaine commun. Elle consiste à juxtaposer les tuples dont la valeur d'un attribut est égal dans les deux tables. C'est une primitive dérivée car elle peut être définie à l'aide des primitives précédentes (exercice laissé au lecteur).

Exemple 6 *CMDCLIENTS = COMMANDES **JOIN** CLIENTS **ON** cmdclt=cltnum*

Les primitives peuvent être combinées pour constituer des requêtes plus élaborées.La séquence d'opérateurs permettant de réaliser une requête élaborée devient assez vite complexe. Le langage SQL permet (heureusement) d'exprimer globalement une requête sans faire apparaître les tables et les primitives intermédiaires. Ce sera le moteur SQL qui sera chargé d'optimiser la requête.

Exemple 7 *une requête SQL qui permet de dresser la liste des noms des clients qui ont acheté des articles de moins de 200F est :*

SELECT DISTINCT cltnom FROM CLIENTS, COMMANDES, LIGCOMMANDES, ARTICLES WHERE cltnum=cmdclt AND artnum=lcdart AND cmdnum=ldccmd AND artpv<200

Une combinaison de primitives permettant d'exécuter cette requête est :

TEMP1 = SELECT ARTICLES WHERE artpv<200
TEMP2 = PROJECT TEMP1 OVER (artnum)
TEMP3 = PROJECT CLIENTS OVER (cltnum,cltnom)
TEMP4 = PROJECT COMMANDES OVER (cmdnum, cmdclt)

TEMP5 = PROJECT LIGCOMMANDES OVER (ldccmd,ldcart)
TEMP6 = TEMP2 JOIN TEMP5 ON artnum=ldcart
TEMP7 = PROJECT TEMP6 OVER (ldccmd)
TEMP8 = TEMP3 JOIN TEMP4 ON cltnum=cmdclt
TEMP9 = PROJECT TEMP8 OVER (cltnom, cmdnum)
TEMP10 = TEMP7 JOIN TEMP9 ON ldccmd=cmdnum
RESULTAT = PROJECT TEMP10 OVER (cltnom)

Une telle séquence n'est, en général, pas unique. La séquence fournie est une des plus efficaces (le lecteur peut s'exercer à en trouver d'autres).

.7 Eléments de SQL

7.1 La commande SELECT

Syntaxe :

SELECT
attributs
expressions extraites
FROM expressions de tables
WHERE conditions de filtrage
GROUP BY critères de regroupement
HAVING conditions de filtrage sur les groupes
ORDER BY critères de tri

Les attributs

permettent de préciser si on souhaite inclure toutes les lignes (attribut **ALL** par défaut) ou seulement les lignes distinctes (attribut **DISTINCT**).
Exemple 8 *SELECT cltloc FROM clients : on affiche les villes des clients avec répétition (il y a autant de lignes affichées que de clients).*

SELECT DISTINCT cltloc FROM clients : on affiche les villes des clients sans répétition.

Les expressions extraites

sont des champs de table, des expressions calculées faisant intervenir les champs, ou des expressions faisant intervenir des fonctions de calcul. Les fonctions utilisées peuvent être des fonctions qui s'appliquent à la valeur d'un champ (upper, year, ...) ou des fonctions de groupe qui effectuent un calcul sur l'ensemble des valeurs d'un champ dans un groupe (max, min, sum, count, avg).
Exemple 9 *SELECT cmdnum, year(cmddate) FROM commandes : on affiche les numéros de commande avec l'année correspondante.*

SELECT cltnom + `` ``+ cltpnom FROM clients : on affiche la concaténation des nom et prénom des clients.

SELECT AVG(artpv) FROM articles : on affiche le prix de vente moyen des articles.

Les expressions de tables

se présentent sous la forme d'une table, d'une jointure entre tables, d'une liste de tables. Les opérateurs de jointure seront : INNER JOIN, LEFT JOIN ou RIGHT JOIN qui correspondent respectivement à la jointure, la jointure externe gauche et la jointure externe droite.

Exemple 10 *SELECT cltnom,cmdnum FROM clients INNER JOIN commandes ON cltnum=cmdclt : on affiche les noms des clients ayant passé une commande avec le numéro de commande correspondant (autant de fois que de commandes).*

SELECT cltnom,cmdnum FROM clients LEFT JOIN commandes ON cltnum=cmdclt : on affiche les noms des clients sans autre information s'ils n'ont pas passé de commandes et avec le numéro de commande correspondant (autant de fois que de commandes) sinon.

Les conditions de filtrage

correspondent à des sélections. On peut utiliser les opérateurs de comparaison usuels. On peut également utiliser les trois opérateurs BETWEEN, LIKE et IN.

remarque Les conditions de jointure peuvent être introduites à l'intérieur de la clause WHERE. Dans ce cas, l'expression de tables correspond à la liste des tables utilisées. L'écriture obtenue est en général plus simple, mais il faut savoir que l'exécution peut être moins efficace. Par exemple, on peut écrire SELECT cltnom, cmdnum FROM clients, commandes WHERE cltnum=cmdclt en lieu et place de SELECT cltnom,cmdnum FROM clients INNER JOIN commandes ON cltnum=cmdclt.

Exemple 11 *SELECT cmdnum FROM commandes WHERE year(cmddate)=1995 : on affiche les numéros de commande de l'année 1995.*

SELECT artnom FROM articles WHERE artpv BETWEEN 100 and 200 : on affiche les noms d'article dont le prix de vente est entre 100 et 200.

SELECT cltnom FROM clients WHERE cltloc IN ('PARIS', 'LILLE') : on affiche les noms des clients parisiens et lillois.

SELECT cltnom FROM clients WHERE cltnom LIKE 'M' : on affiche la liste des clients dont le nom commence par M.*

Les critères de regroupement.

Un groupe est un sous-ensemble des tuples d'une table (qui peut être le résultat d'une requête), pour lesquels les valeurs du champ de regroupement reste constante. Les groupes sont spécifiés par la clause GROUP BY suivie du nom du champ sur lequel on effectue le regroupement. Les fonctions de groupe usuelles sont : MAX, MIN, SUM, COUNT, AVG. La clause HAVING est l'équivalent du WHERE appliqué aux groupes. Les conditions de filtrage de la clause HAVING sont des conditions qui portent sur les fonctions de groupe. Les critères de la clause WHERE s'appliquent aux tuples, ceux de la clause HAVING s'appliquent aux groupes. Le moteur SQL élimine d'abord les tuples qui ne satisfont pas aux conditions du WHERE, puis il constitue les groupes et leur applique les conditions du HAVING.

Exemple 12 *SELECT artnom, SUM(ldcqte) FROM (articles JOIN ligcommandes ON artnum=ldcart) JOIN commandes ON ldccmd=cmdnum WHERE YEAR(cmddate)=1995 GROUP BY artnom HAVING SUM(ldcqte)>100 : on affiche les noms d'articles avec la quantité totale commandée pour les articles commandés en plus de 100 exemplaires dans l'année 1995.*

Les critères de tri
(ascendant ou descendant) sont des noms de champs extraits, ou d'expressions extraites.

.7.2 composition de requêtes

La clause UNION
permet de réaliser la réunion du résultat de deux requêtes. Il faut que les structures sélectionnées soient compatibles.
Exemple 13 *SELECT artnom FROM articles WHERE artpv>200 UNION SELECT artnom FROM articles JOIN ligcommandes ON artnum=ldcart WHERE ldcqte>10 : on affiche les noms d'article de prix de vente supérieur à 200F ou qui ont été commandé dans une commande en plus de 10 exemplaires.*

Imbrication de requêtes.
On peut imbriquer les requêtes. Les conditions de filtrage peuvent donc porter sur le résultat de sous-requêtes.

- La requête intérieure renvoie une valeur. Cette valeur peut être utilisée comme valeur de comparaison dans une clause WHERE.

  Exemple 14 *SELECT artnom,artpv FROM articles WHERE artpv < (SELECT artpv FROM articles WHERE artnum='A0056') : on affiche les noms et prix d'articles de prix de vente inférieur au prix de l'article de numéro A0056.*

- La requête intérieure renvoie un ensemble de valeurs. Elle peut être alors utilisée dans une clause WHERE comportant l'un des opérateurs IN, ANY, ALL et EXISTS. La condition expression IN sous-requête teste l'appartenance à un ensemble de valeurs fournies par la sous-requête qui ne doit comporter qu'un seul champ extrait. La condition expression > ANY sous-requête teste si l'expression est supérieure à au moins un élément de l'ensemble de valeurs fournies par la sous-requête qui ne doit comporter qu'un seul champ extrait. La condition expression > ALL sous-requête teste si l'expression est supérieure à tous les éléments de l'ensemble de valeurs fournies par la sous-requête qui ne doit comporter qu'un seul champ extrait. La condition EXISTS sous-requête a la valeur vrai si la sous-requête extrait un ensemble non vide de valeurs.

  Exemple 15 *SELECT artnom FROM articles WHERE artpv>ANY (SELECT artpv FROM articles WHERE artcoul='blanc') : on affiche les noms d'article de prix de vente supérieur au prix d'au moins un article de couleur blanche.*

.7.3 Gestion des transactions en SQL

Un SGBDR est constitué des éléments fondamentaux suivants :

Un moteur relationnel
: responsable de toutes les opérations sur les données.

Un dictionnaire
: contenant la description de toutes les données de la base.

Un système de gestion des transactions
: qui assure que les données restent intègres (ou cohérentes) quels que soient les évènements qui peuvent intervenir. Le but de ce paragraphe est de décrire succinctement le problème et les solutions apportées en SQL.

Une application est constituée d'une succession de groupes d'opérations. Chaque groupe forme une unité logique de travail qui est indivisible : c'est-à-dire toutes les opérations du groupe sont effectuées ou aucune ne l'est.

Exemple 16 *En comptabilité, si une somme doit transiter d'un compte vers un autre, les deux opérations de mise à jour (crédit et débit) doivent s'effectuer toutes les deux ou ne pas s'effectuer du tout pour que les données restent cohérentes.*

Quel que soit le système informatique utilisé, divers éléments peuvent perturber l'exécution d'une séquence d'instructions. Parmi ceux-ci, on peut citer : une erreur de programmation, une panne, l'exécution d'une tâche plus prioritaire. Ce dernier point sera très fréquent dans un environnement client-serveur où le nombre de tâches concurrentes est très élevé et dans lequel un client peut, à tout moment, demander l'exécution d'une application.

Le mécanisme des transactions résout les problèmes dûs aux accès concurrents. Une transaction est un groupe d'opérations tel que toutes les opérations sont effectuées ou aucune ne l'est. Une transaction est vue de l'extérieur comme une opération atomique (c'est à dire une seule opération). Chaque transaction s'effectue indépendamment des autres (tout se passe comme si les transactions étaient exécutées l'une après l'autre). SQL peut gérer les transactions grace aux deux instructions COMMIT et ROLLBACK. L'instruction COMMIT demande au gestionnaire de transactions de rendre les modifications permanentes (aucun problème n'a été rencontré). L'instruction ROLLBACK signale la fin prématurée de la transaction et demande au gestionnaire de transactions de défaire tout ce qui a été fait depuis le début de la transaction, c'est-à-dire de retourner dans l'état cohérent correspondant au début de la transaction.

Chaque instruction SQL forme en soi une transaction. Il n'y a pas d'instruction particulière pour préciser le début d'une transaction. Une transaction démarre chaque fois qu'on exécute une instruction. Par contre, une transaction est cloturée explicitement par les instructions ROLLBACK ou COMMIT. La norme SQL définit deux caractéristiques pour la transaction : son mode et son niveau d'utilisation. Le mode détermine les opérations possibles dans la transaction : READ ONLY (lecture seule) et READ WRITE (lecture et écriture). Le niveau d'isolation indique le comportement de la transaction par rapport aux autres transactions.

.7.4 Contrôle de l'accès aux données

Cette partie concerne essentiellement l'administrateur de la base de données, mais un utilisateur intelligent doit posséder quelques notions sur les privilèges. Le principe fondamental est qu'un utilisateur ne peut effectuer une opération quelconque que s'il détient le privilège approprié pour cette opération. C'est l'administrateur qui donne (ou retire) les privilèges grace aux instructions GRANT et REVOKE.

Les privilèges

SELECT
: privilège donnant accès à toutes les colonnes d'une table.

INSERT
: permet l'insertion dans une table.

UPDATE(colonnes)
: permet la modification de toutes les colonnes d'une table. En spécifiant certaines colonnes de la table, le privilège est limité à celles-ci.

DELETE
: permet la suppression des lignes d'une table.

ALTER
: pour modifier la structure d'une table.

INDEX
: pour gérer les index d'une table.

ALL
: tous les droits.

Les instructions GRANT et REVOKE

`GRANT privilèges ON objet TO liste_utilisateurs`
: donner les privilèges sur les objets (tables, vues) à certains utilisateurs.

`REVOKE privilèges ON objet FROM utilisateurs`
: retirer les privilèges.

Exemple 17

Permettre à tout le monde de lire la table fournisseurs :

```
GRANT SELECT ON fournisseurs TO PUBLIC
```

Permettre à Fernand de mettre à jour les attributs lcdpdt et lcdqte de ligcommande :

```
GRANT UPDATE(lcdpdt,lcdqte) ON ligcommande TO fernand
```

Donner tous les privilèges à Sophie sur la table clients sauf la suppression de lignes :

```
GRANT ALL ON clients to sophie
REVOKE DELETE ON clients to sophie
```

En combinant la notion de vue (partie visible de la base de données fournie à un utilisateur) et une gestion adéquate des privilèges, on atteint une grande souplesse de manoeuvre pour sécuriser les données.

.8 Exercices

Dans tous les exercices, nous utilisons la base `Bibliothèque`. Le schéma relationnel est donnée dans la figure suivante :

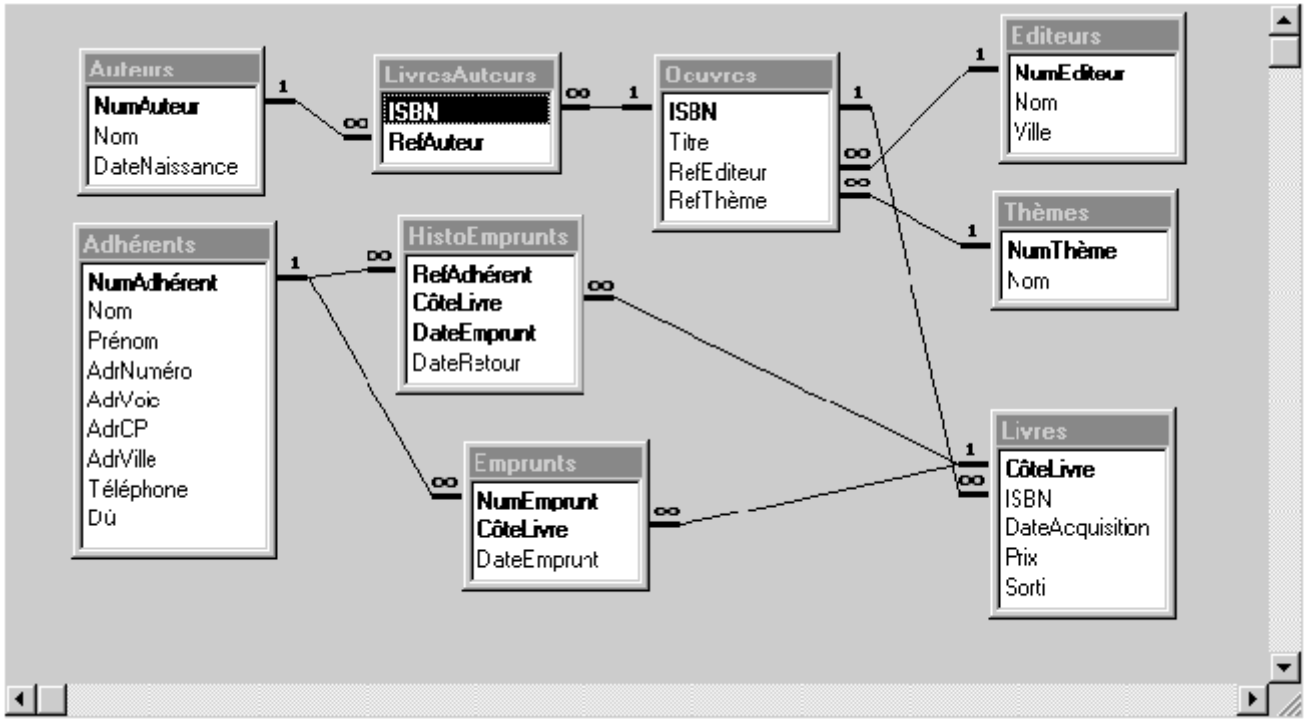

Figure 3 : Schéma relationnel de la base `Bibliothèque`.

.8.1 SQL et QBE

Nous allons voir les deux façons de définir des requêtes dans les SGBDR. La première consiste en l'utilisation du *QBE (Query By Example)* qui consiste en l'utilisation d'une interface graphique permettant la définition des requêtes. La seconde consiste en l'utilisation d'un langage d'interrogation *SQL (Standard Query Language).*

Comme vous ne connaissez pas (encore) SQL, nous allons définir des requêtes à l'aide du QBE d'ACCESS, puis nous visualiserons la traduction de chacune des requêtes en SQL.

1. Lancez Access et ouvrez la base `Bibliothèque`.
2. Définir une requête qui affiche tous les titres de livre avec le nom d'éditeur. Voir le SQL. modifier la définition pour trier par éditeur et par titre. Voir le SQL. Retirer le ORDER BY dans la requête SQL. Voir la modification dans le QBE.
3. Définir une requête qui affiche les titres de chez **Dunod**. Voir le SQL.
4. Définir une requête qui affiche la liste des auteurs ayant écrit une oeuvre éditée chez **Dunod**. Avec et sans répétition. Voir le SQL.
5. Définir une requête qui compte le nombre d'oeuvres par éditeur. Voir le SQL.
6. Définir une requête qui compte le nombre de livres par éditeur. Voir le SQL.

.8.2 Écriture de requêtes SQL

1. Regarder l'aide sur l'instruction

```
2.          SELECT (SQL)
3.          Attributs ALL,...
```

```
4.          clause FROM
5.          clause WHERE
```

6. Pour définir une requête SQL choisir requête,nouveau, ne pas ouvrir de table, puis cliquer sur le bouton SQL de la barre d'outils. Vous êtes dans une fenêtre d'édition dans laquelle nous allons taper une première requête SQL.

```
7.     SELECT all oeuvres.titre FROM livres, thèmes, oeuvres WHERE
8.     oeuvres.refthème=thèmes.numthème and livres.isbn=oeuvres.isbn and
9.     thèmes.nom = "FINANCES" ORDER BY titre;
```

10. Exécutez cette requête (icône feuille de données de la barre d'outils) , la comprendre. L'exécuter avec l'attribut distinct, puis avec l'attribut `distinctrow`.

11. Écrire une requête qui affiche les titres des livres disponibles. On utilisera le champ Sorti de la table livres. On donnera les deux façons : sans répétition et avec autant de répétitions qu'il y a d'exemplaires disponibles.

12. Afficher les côte et titre des livres achetés avant 1970.

13. Consulter l'aide sur la clause GROUP BY. Écrire une requête SQL qui affiche pour chaque oeuvre donnée par son titre, le nombre d'exemplaires présents et le prix d'achat moyen de cet ouvrage.

14. Requêtes imbriquées :
    a. Calculer le prix d'achat moyen de tous les livres de la bibliothèque.
    b. Afficher les côtes des livres dont le prix est supérieur au prix moyen.

Requêtes avancées

1. Écrire une requête qui affiche le nombre d'emprunts par client.

2. Même question avec l'historique.

3. Et le nombre total d'emprunts ? créer une requête union, la sauvegarder, utiliser cette requête dans une requête de calcul.

4. Écrire une requête qui affiche la liste des clients avec les côtes des livres empruntés (emprunts en cours).

5. Modifier la requête pour afficher tous les clients avec les côtes des livres empruntés et rien si ils n'ont pas d'emprunt en cours.

6. Écrire une requête qui liste les adhérents donnés par leur numéro, nom et prénom concaténés, avec les côtes des livres empruntés (emprunts en cours), la date d'emprunt (date d'emprunt, et le titre. Définir un formulaire colonne basé sur cette requête.

7. Écrire une requête qui liste les clients donnés par leur nom et prénom, avec les cotes des livres empruntés (emprunts en cours), la date de retour maximum (date d'emprunt+ 21), et le titre.

8. Définir la requête SQL suivante :

```
9.       UPDATE DISTINCTROW EMPRUNTS
10.         SET EMPRUNTS.DATEEMPRUNT = EMPRUNTS.DATEEMPRUNT+1080
```

Cette requête est une requête de mise a jour (on met a jour les dates d'emprunt car la base est âgée). Visualisez la feuille de données : bouton feuille de données. Puis, l'exécuter : bouton ! de la barre d'outils. Remarquez le message de mise en garde. Visualisez la feuille de données.