afpa
La formation professionnelle

Secteur Tertiaire Informatique
Filière étude - développement

Introduction aux bases de données et au langage SQL

Accueil

Apprentissage

Période en entreprise

Evaluation

Afpa Créteil

1 LES « MODÈLES » DE BASES DE DONNÉES

Les bases de données existent depuis longtemps en informatique. Leur but est de **représenter** le monde réel à l'aide de moyens informatiques performants et offrant un bon niveau de **sécurité** et d'**intégrité**.
Avant cette notion de bases de données, chaque programme gérait ses propres fichiers de données, sans lien avec les autres applications. Ceci avait pour inconvénients essentiels :

- la **ressaisie** (ou, au mieux, la récupération) de données d'une application à l'autre, au détriment des performances,
- la **redondance** (= la duplication) des données, au détriment de la sécurité et de l'intégrité.

Les représentations informatiques des bases de données se sont appuyées sur différentes théories mathématiques ; elles ont abouti à 4 « modèles » principaux :

- le modèle **hiérarchique**,
- le modèle **réseau**,
- le modèle **relationnel (**1970, E.F.Codd),
- le modèle **objet**.

Nous allons en décrire brièvement les principes. Seul le modèle relationnel est effectivement mis en œuvre (et de plus en plus) dans les Système de Gestion de Bases de Données (SGBD).

1.1 LES COMPOSANTES D'UN SGBD

Un SGBD [1] doit offrir aux développeurs des fonctions de **définition de la Base de données** et de **manipulation des données**. Il propose donc un **langage de définition des données (LDD[2])** et (au moins) un **langage de manipulation des données (LMD[3])** souvent appelé **langage de requêtes**. Il peut aussi offrir un **langage de contrôle de données (LCD[4])** qui permet de créer des utilisateurs et de leur attribuer des privilèges (autorisation de mise à jour, de suppression, de création, …) et un **langage de contrôle de transaction (LCT[5])**

Un SGBD doit être « interfacé » avec les programmes d'applications : il doit offrir des liens avec les langages de programmation. Par exemple, on peut intégrer des requêtes SQL dans un programme PhP. Certains SGBD proposent des fonctions de « compilation de requêtes ».

Un SGBD est conçu pour exploiter le système de gestion physique de fichiers du système d'exploitation de la machine : au bout du compte, quelle que soit l'organisation des données décrite dans la base, il faut manipuler des fichiers physiques enregistrés sur des supports magnétiques ; **le SGBD ne se substitue pas au système d'exploitation, il l'utilise**.

---

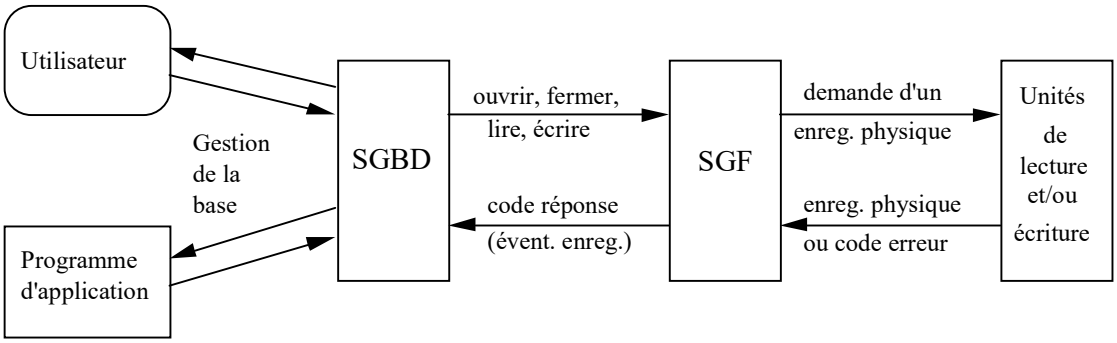
[1] SGBD : System de Gestion de Base de Données (en anglais *DBMS* pour *database management system*)

[2] LDD : Langage de Description de Données (*en anglais data definition language*, DDL)

[3] LMD : Langage de Manipulation de Données (*en anglais data manipulation language*, DML)

[4] LCD : Langage de Contrôle de Données (*en anglais data control language*, DCL)

[5] LCT : Langage de Contrôle de Transaction *(en anglais Transaction Control Language, TCL)*



Les principales fonctions offertes par un SGBD sont :

- un langage de **définition des données: LDD**,
- un langage de **manipulation des données: LMD**,
- un langage de **contrôle de données** sur la base : **LCD**,
- un langage de **contrôle des transactions** : **LCT**
- un **gestionnaire de la base de données** (cœur du SGBD, définition des opérations élémentaires à effectuer),
- une **interface avec le système d'exploitation** de la machine,
- une **interface avec les langages de programmation**.

1.2 LES NOTIONS D' « ENTITÉ » ET D' « ASSOCIATION »

Une **représentation abstraite** couramment utilisée dans le domaine des bases de données est le « **modèle entité–association** ». Essayons d'en définir les principes de base de manière à cerner les principales différences entre les trois modèles : hiérarchique, réseau et relationnel. Une *entité* correspond à une **information ayant une existence propre et une importance pour la gestion de l'entreprise**. C'est, par exemple, un client, un article de catalogue, un fournisseur, un salarié ... Une entité est définie par un ensemble de **« propriétés »** (ou « **attributs** » ou « champs ») qui la constitue (la Raison Sociale, le nom du salarié, le prix de vente hors taxes, le salaire mensuel...).

Une association correspond à un **lien logique entre entités** (deux, en général). Une association peut « porter » ou non des attributs. Par exemple, l'association « commande » entre les entités « article » et « client » peut porter les propriétés « quantité », « prix de vente », « date de commande » ... L'association « est chef de » qui est « réflexive » sur l'entité « salarié » n'est pas porteuse de propriété.

Une caractéristique des associations est le **nombre d'associations possibles entre entités**. Un client peut commander plusieurs produits et chaque produit peut être commandé par plusieurs clients : l'association « commande » est de type « **m-n** ». Dans le cas de l'association « est chef de », un salarié n'ayant qu'un seul supérieur hiérarchique direct, elle est qualifiée de relation « **n-1** » ou « **1-n** ». On peut aussi rencontrer des associations de type « **1-1** » (chaque représentant couvre un secteur commercial spécifique).

1.3 LE MODÈLE HIÉRARCHIQUE

Le *modèle hiérarchique* est le plus ancien. La réalisation la plus connue et la plus utilisée est IMS, pour gros systèmes IBM (1972). Son langage de manipulation de données porte le nom DL/1.

Ce modèle est tout à fait **adapté à la représentation des relations de type « 1-1 » et « 1-n »**. Chaque entité peut ainsi aisément être reliée à une ou plusieurs autres (les commandes d'un client, les subordonnés d'un chef, un organigramme d'entreprise...). Par contre, il ne propose pas de représentation directe des relations de type « m-n » (pourtant très courantes) ; l'artifice utilisé, pour éviter les redondances de données, est de créer des articles «virtuels» ce qui complique un peu les choses et nécessite que le programmeur compense en quelque sorte les lacunes du système.

1.4 LE MODÈLE RÉSEAU

Le *modèle réseau* a été normalisé par le DBTG (Data Base Task Group) de la CODASYL (Conference On DAta SYstems Languages - années 1970). Il peut être vu comme une évolution du modèle hiérarchique : **il permet de représenter les relations de type «m-n»**. Par contre, les relations réflexives nécessitent toujours la définition d'articles ou champs virtuels (personne n'est parfait).
Le modèle réseau offre tout un environnement de programmation qui facilite la vie du programmeur.

1.5 LE MODÈLE RELATIONNEL

Le *modèle relationnel* n'utilise que le concept de relation du modèle entité-relation. Il repose sur le cadre mathématique de « l'algèbre relationnelle ». L'américain E.F. CODD, chercheur chez IBM et « père » incontesté du modèle, a identifié 12 règles permettant de définir le modèle.
Ce modèle repose sur le concept de « **table** », encore appelée « **relation** », composée de « **lignes** » (ou « **tuples** ») et de « **colonnes** » (ou « **attributs** »).

Le parallèle avec les fichiers de données classiques est aisé : une table peut être matérialisée par un fichier de données, une ligne par un enregistrement et une colonne par un champ.

Chaque colonne (représentant un attribut) est associée à un **type** ; les valeurs prises par les différentes lignes pour cette colonne doivent appartenir à un même **domaine** (nombres entiers, réels, chaînes de caractères...).

Exemple : représentation d'une table du personnel

Matricule	Nom	Poste	Salaire	N° dept
350	Durand	Employé	8000	320
780	Dupond	Cadre	15000	870
320	Veillon	PDG	25000	400
490	Martin	Cadre	15000	320

Le « **schéma de la relation** », composé de son nom suivi de la liste de ses attributs, définit parfaitement une table. Par exemple, la table représentant les voitures et composée de leur nom, type, kilométrage et année sera définit par :

```
VOITURE (MODELE : CHAR(20),
         TYPE : CHAR(10),
         KILOMETRAGE : ENTIER,
         ANNEE : ENTIER)
```

Il est d'usage de souligner le ou les attributs constituant la « **clé** » ou « **critère d'identification** » des lignes. L'ensemble des schémas des relations constitue le **schéma de la Base de données**.

Le passage du modèle entité–relation au modèle relationnel se fait en respectant un certain nombre de règles ; on parle de « formes normales » (« première », « deuxième » et « troisième forme normale » - ou «1FN», «2FN» et «3FN» - et «forme normale de Boyce-Codd»). Une littérature abondante traite de toutes ces notions théoriques ; notons qu'il est difficile de trouver des ouvrages simples, clairs et pas trop mathématiques sur ce domaine des bases de données.

Commercialement, les SGBD relationnels (ou « SGBDR ») sont en pleine expansion, sur toutes les gammes de matériels (ACCES, DB2, INGRES, ORACLE, PARADOX, MySQL). Le support du langage de manipulation de données standard SQL, bien adapté à ce modèle, participe au succès du modèle relationnel.

2 NOTIONS D'ALGÈBRE RELATIONNELLE

La manipulation des bases de données conformes au modèle relationnel est très proche de la manipulation des tables selon les lois de l'algèbre relationnelle. En conséquence, il est important de bien « visualiser » les concepts véhiculés par une commande SQL afin de mieux comprendre le sens des mots-clés et des options offertes... ainsi que les inévitables erreurs de manipulation du langage.

Un bref tour d'horizon de la théorie sous-jacente est donc nécessaire.

2.1 À LA BASE, LA THÉORIE DES ENSEMBLES

L'algèbre relationnel manipule des **ensembles** (au sens mathématique) de données ; en conséquence :

- l'ordre des lignes n'a pas d'importance (**pas de notion de tri**),
- chaque « tuple » est unique dans une même table (**pas de redondance des lignes possible**).

Les opérations de base proposées par cette algèbre portent :

- sur les tables (**opérations «binaires»** mettant en œuvre deux tables) comme l'union, la différence, l'intersection ou le produit cartésien ;
- sur les lignes d'une même table (**opérations «unaires»**) comme la projection ou la restriction.

A ces opérations de base sont ajoutées des opérations plus complexes (jointures) qui sont en fait des compositions d'opérations élémentaires (tout comme la puissance arithmétique revient à un ensemble de multiplications).

Le résultat de toute manipulation algébrique est une nouvelle table.

2.2 LES OPÉRATIONS « BINAIRES » DE BASE

2.2.1 L'union

L'*union* n'est possible que pour deux tables de **même schéma relationnel** (même structure).

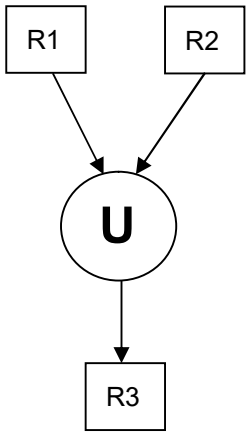
Elle forme une nouvelle table composée de l'**ensemble des lignes issues de la première et de la deuxième table**. Si une même ligne existe dans les deux tables, une seule ligne sera insérée dans la table résultante (élimination des « doublons »).

Exemple :

VINS1	Cru	Millésime	Région	Couleur
	Chenas	1978	Beaujolais	Rouge
	Tavel	1979	Rhône	Blanc
	Tokay	1980	Alsace	Rosé

VINS2	Cru	Millésime	Région	Couleur
	Chablis	1985	Bourgogne	Rouge
	Tokay	1980	Alsace	Rosé

VINS3	Cru	Millésime	Région	Couleur
	Chenas	1978	Beaujolais	Rouge
	Chablis	1985	Bourgogne	Rouge
	Tavel	1979	Rhône	Blanc
	Tokay	1980	Alsace	Rosé



2.2.2 La différence

La *différence* n'est possible que pour deux tables de même schéma. La table résultante est formée des **lignes qui appartiennent à la première table mais pas à la seconde**.

Il apparaît clairement que, dans cette opération, l'ordre des opérandes est important! La différence est un opérateur non commutatif.

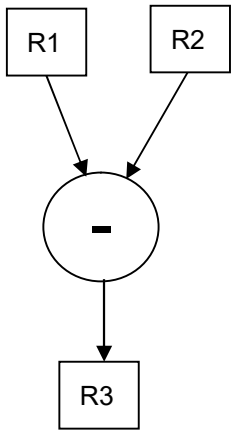
Exemple:

VINS1	Cru	Millésime	Région	Couleur
	Chenas	1978	Beaujolais	Rouge
	Tavel	1979	Rhône	Blanc
	Tokay	1980	Alsace	Rosé

-

VINS2	Cru	Millésime	Région	Couleur
	Chablis	1985	Bourgogne	Rouge
	Tokay	1980	Alsace	Rosé

VINS	Cru	Millésime	Région	Couleur
	Chenas	1978	Beaujolais	Rouge
	Tavel	1979	Rhône	Blanc



2.2.3 L'intersection

L'*intersection* n'est possible que pour deux tables de même schéma.
La table résultante est formée des **lignes qui appartiennent à la fois à la première et à la seconde table**.

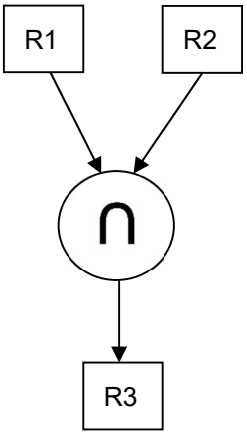
VINS1	Cru	Millésime	Région	Couleur
	Chenas	1978	Beaujolais	Rouge
	Tavel	1979	Rhône	Blanc
	Tokay	1980	Alsace	Rosé

-

VINS2	Cru	Millésime	Région	Couleur
	Chablis	1985	Bourgogne	Rouge
	Tokay	1980	Alsace	Rosé

⇩

VINS	Cru	Millésime	Région	Couleur
	Tokay	1980	Alsace	Rosé



2.2.4 Le produit cartésien

Le *produit cartésien* peut être effectué sur deux tables quelconques.
Le **schéma résultant** est constitué de la **réunion des schémas** des tables mises en œuvre (ensemble des colonnes). **Chaque ligne de la première table est combinée avec toutes les lignes de la seconde**.
Le nombre de lignes résultant est donc le produit des nombres de lignes de chacune des tables (le produit cartésien de deux tables de 1000 tuples représente une table d'un million de lignes !). Le résultat d'un produit cartésien peut donc être rapidement très volumineux (et long à établir). Un SGBD doit éviter au maximum d'effectuer *réellement* des produits cartésiens entre tables. Le degré d'optimisation des requêtes est justement un critère de qualité des SGBD.

Exemple:

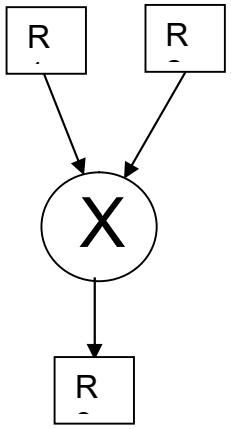
VINS1	Cru	Région	Couleur
	Chenas	Beaujolais	Rouge
	Tokay	Alsace	Rosé

X

ANNEES	Millésime
	1985
	1980

⇩

VINS	Cru	Millésime	Région	Couleur
	Chenas	1980	Beaujolais	Rouge
	Tokay	1980	Rhône	Rosé
	Chenas	1985	Beaujolais	Rouge
	Tokay	1985	Rhône	Rosé



2.2.5 Exercices sur les tables

Effectuez l'union, la différence, l'intersection des deux tables VOITURE1 et VOITURE2 ci-après :

Table VOITURE1

MODELE	TYPE	KILOMETRAGE	ANNEE
NEVADA	BREAK	25800	1992
ESCORT	BERLINE	130580	1986
TRANSPORTER	UTILITAIRE	88900	1987

4L	BERLINE	153800	1970
2CV	BERLINE	156000	1953

Table VOITURE2

MODELE	TYPE	KILOMETRAGE	ANNEE
NEVADA	BREAK	25800	1992
405	BERLINE	44600	1989
2CV	BERLINE	156000	1953
ESCORT	BERLINE	20580	1992

Effectuez le produit cartésien des deux tables VOITURE1 et CHAUFFEUR.

Table CHAUFFEUR

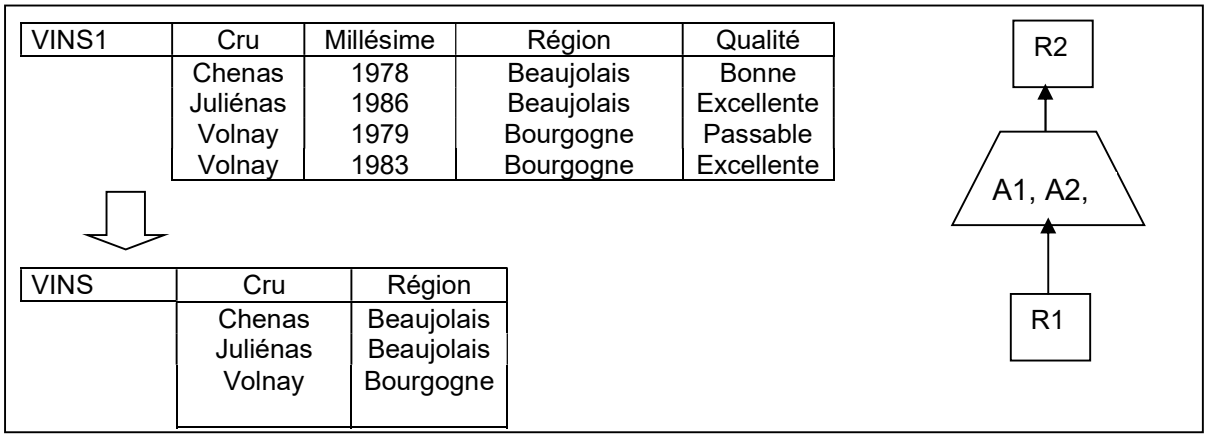
NOM	PERMIS
OLLIER	B
MONTHERLAND	A, B
LEFEBVRE	B, C1

2.3 LES OPÉRATIONS «UNAIRES» DE BASE

2.3.1 La projection

La *projection* permet de sélectionner certaines **colonnes** d'une table afin de constituer une nouvelle relation. Il faut souligner que l'élimination des doublons est nécessaire. En effet, après suppression de certains attributs, deux lignes peuvent devenir complètement identiques.

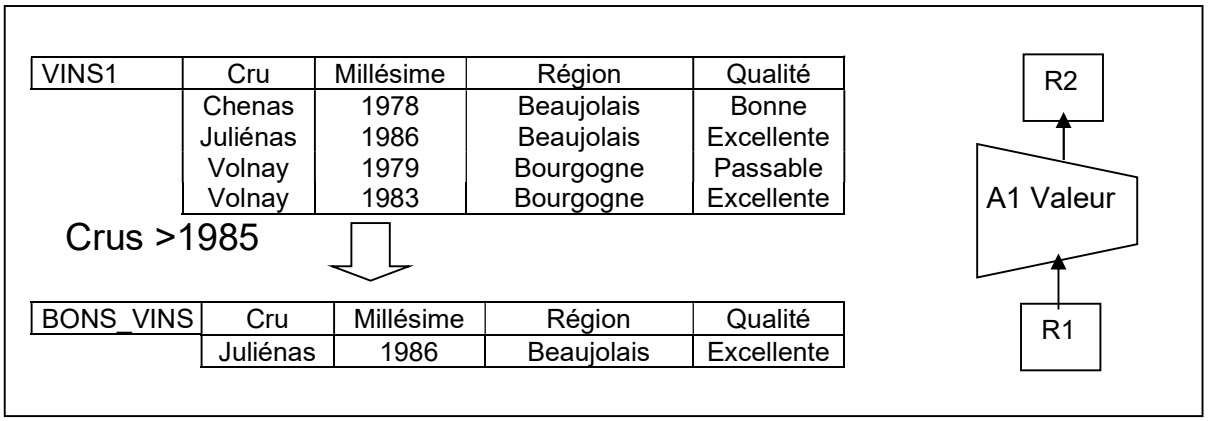
Exemple:



2.3.2 La sélection ou restriction

La *restriction* permet de sélectionner certaines **lignes** d'une table afin de constituer une nouvelle relation.
La sélection des lignes s'effectue suivant une **condition** exprimée sous forme d'une **expression logique appliquée à chaque tuple** et incluant des opérateurs de comparaison (=, >, >=...) et des opérateurs logiques (ET, OU...).
Application : liste des clients parisiens dont le chiffre d'affaires est supérieur à 100 000 € (ici, on est bien en présence de 2 conditions combinées par un «ET» logique, «chiffres d'affaires > 100 000 €», «région = "PARIS"»).

Exemple: La restriction de la relation VINS consistant à créer une relation BONS_VINS par la condition Qualité = "Excellente" et millésime >1985 fournit le résultat :



Les opérations élémentaires sur les tables décrites ci-dessus peuvent se combiner afin d'aboutir à un résultat plus fin. Sur une même table, **projection et restriction** sont le plus souvent associées, par exemple listage des seules rubriques «Raison Sociale» et «chiffre d'affaires» des clients parisiens dont le chiffre d'affaires est supérieur à 100 000 F (on retrouve bien la projection des rubriques en plus de la sélection des lignes).

2.3.3 LES JOINTURES

Il est courant qu'une même interrogation (ou «requête») concerne des données issues de plusieurs tables. Il est alors nécessaire d'effectuer le produit cartésien des tables mises en œuvre. L'inconvénient du produit cartésien est qu'il génère **toutes les combinaisons possibles** entre les lignes des tables, alors que bon nombre d'entre elles n'ont aucune signification dans le réel.
Par exemple, le produit cartésien entre la table des clients et celle des représentants va générer une ligne par combinaison possible client/représentant, sans se préoccuper de savoir qu'un client dépend d'un représentant et d'un seul.
Après exécution du produit cartésien, une sélection est donc nécessaire afin de ne conserver que les lignes ayant une existence logique.
Dans notre exemple, on ne gardera que les lignes combinant chaque client avec le représentant associé. Cette existence logique est matérialisée par une **information commune dans les deux tables**, en général, un attribut. Ici, on identifiera chaque représentant par un code (attribut–clé pour cette table) et on mémorisera le code du représentant concerné dans la table des clients.

Cette «**restriction de concordance**», due au produit cartésien, s'exprime par un critère comparant les attributs communs.

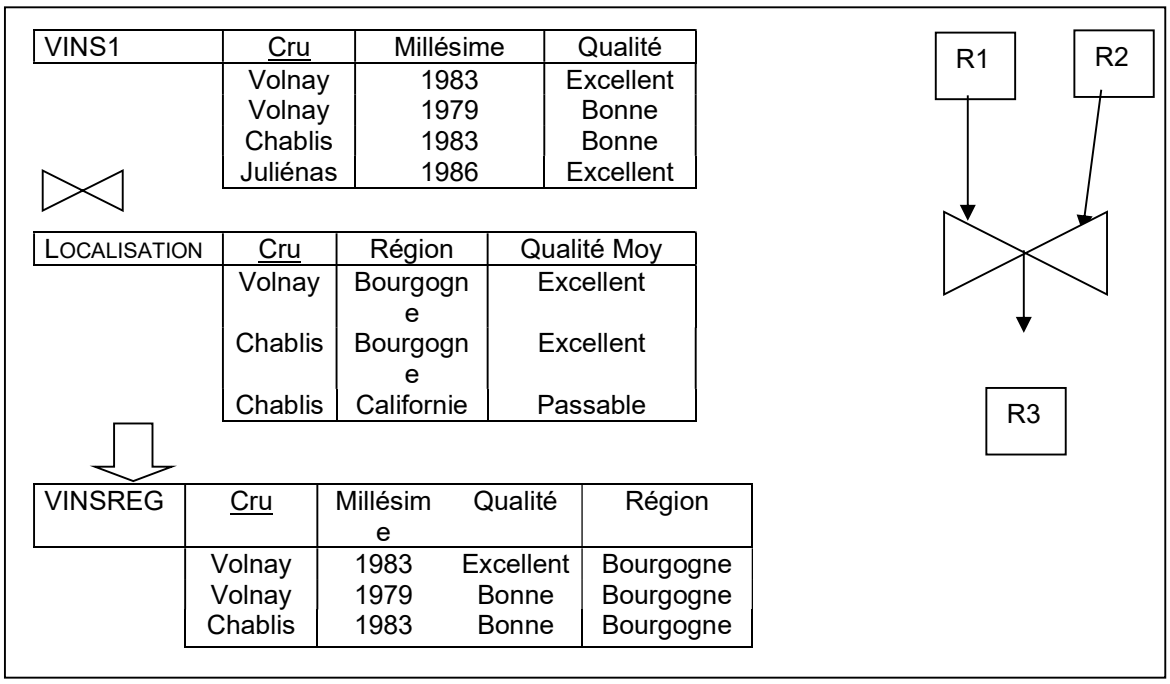
En plus de ces deux opérations (produit cartésien et restriction), on effectue la plupart du temps d'autres sélections (afin de se focaliser sur certaines lignes de la table résultante) et une projection (afin de ne conserver que les colonnes significatives pour le résultat souhaité).

La combinaison des 3 opérations produit cartésien, restriction et projection constitue ce que l'on appelle une «jointure».

La littérature distingue habituellement plusieurs type de jointures selon l'expression de la condition de concordance ou selon la projection réalisée. Ainsi, l'«équi-jointure» utilise une condition d'égalité sur les attributs communs (l'exemple détaillé ci-dessus en est une illustration), la «semi–jointure» effectue une projection sur les seules colonnes de la première table (par exemple, la recherche des clients dont le représentant a dépassé l'objectif annuel met bien en œuvre les tables «client» et «représentant» mais le résultat voulu ne concerne que les informations des clients).

Les requêtes SQL de base permettent d'effectuer des jointures.

Exemple: Vin de Bourgogne avec millésime de qualité bonne ou excellente.



2.3.4 Exercice sur les jointures.

Décrivez les opérations de base nécessaires pour obtenir le résultat de la requête ci-dessous. Effectuez manuellement la jointure correspondante.

Table VOITURE

MODELE	TYPE	KILOMETRAGE	ANNEE	Chauffeur
NEVADA	BREAK	25800	1992	10
ESCORT	BERLINE	130580	1986	11
TRANSPORTER	UTILITAIRE	88900	1987	10
4L	BERLINE	153800	1970	12
2 CV	BERLINE	156000	1953	11

Table CHAUFFEUR

NOM	PERMIS	CODE
OLLIER	B	10
MONTHERLAND	A,B	12
LEFEBVRE	B,C1	11

Requête
«Modèle, kilométrage et année des véhicules conduits par le chauffeur "LEFEBVRE"».

2.4 REGLES OU CONTRAINTE D'INTEGRITE

Les règles d'intégrité sont les assertions que doivent vérifier toutes les données stockées dans une base. Certaines de ces contraintes sont qualifiées de "structurelles" : elles sont inhérentes au modèle, c'est-à-dire nécessaires à sa mise en œuvre, à son bon fonctionnement. D'autres peuvent apparaître comme étant des règles de "comportement", propres au schéma particulier d'un système d'information, d'un type d'applications.

En fait, le modèle relationnel impose à priori une règle structurelle, l'unicité des clés. Il est commode, et courant, d'y ajouter deux types de règles d'intégrité supplémentaires - les contraintes de référence et les contraintes d'entité - ceci afin d'obtenir les règles d'intégrité minimum supportées par le modèle relationnel.

2.4.1 unicité de clé

Par définition, une relation est un ensemble de tuples. Un ensemble n'ayant pas d'élément en double, il ne peut donc exister deux fois le même tuple dans une relation. Afin d'identifier les tuples d'une relation sans en fournir toutes les valeurs, la notion de clé est utilisée.

> **La clé est l'ensemble d'attributs minimum dont la connaissance des valeurs permet d'identifier, de manière unique, un tuple de la relation considérée.**

Toute relation doit posséder au moins une clé. Dans le cas où il en existe plusieurs, on en choisit généralement une de manière arbitraire, qui est appelée **clé primaire** (PRIMARY KEY). Il est à noter que la détermination d'une clé au sein de la relation nécessite une réflexion importante : il s'agit de prévoir toutes les extensions possibles de la relation et de s'assurer que l'ensemble des attributs retenus pour constituer la clé permettra toujours l'identification d'un tuple unique !

2.4.2 intégrité référentielle

En termes de base de données, une entité correspond à un tuple dans une relation, qui comporte la clé de l'entité (identifiant) et ses caractéristiques (sous forme d'attributs). Une association est également modélisée par une relation : celle-ci doit comporter les clés des entités participantes (pour assurer son unicité) ainsi que les caractéristiques propres à l'association.

On distingue en fait deux catégories de relations : les relations indépendantes ou statiques (matérialisant des entités) et les relations dépendantes ou dynamiques (matérialisant des associations).
Cette dépendance est caractérisée par la présence de **clés étrangères** (FOREIGN KEY), c'est-à-dire d'attributs qui sont clés primaires dans d'autres relations.

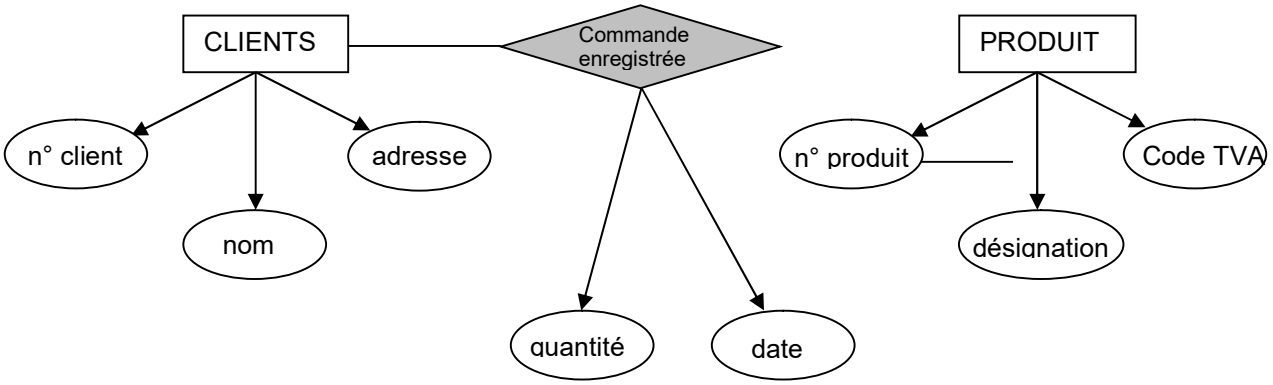
> **La contrainte de référence est une contrainte d'intégrité qui, portant sur une relation R1, consiste à imposer que la valeur d'un groupe d'attributs de R1 apparaisse comme valeur de clé primaire dans une (ou plusieurs) autre (s) relation (s).**

Exemple: Considérons un système d'information partielle concernant la gestion commerciale d'une entreprise. On peut imaginer les relations suivantes :

CLIENT [n° client, nom, adresse,...]
PRODUIT [n° produit, désignation, code TVA,..]

et la relation COMMANDE-ENREGISTREE

[n° client, n° produit, quantité, date,...] où les attributs n° client et n° produit sont des clés étrangères.



Cette intégrité référentielle doit induire certains contrôles automatiques de la part d'un SGBD relationnel. Ainsi, on ne peut pas mémoriser, dans la relation COMMANDE_ENREGISTREE, de tuple comportant un n° client inexistant dans la relation CLIENT ou un n° produit inconnu dans la relation PRODUIT.

2.4.3 intégrité d'entité (ou de relation)

Lors de l'insertion de tuples dans une relation, il arrive fréquemment qu'un attribut soit inconnu, ou vide de sens, sous certaines conditions; on est alors conduit à introduire dans la relation une valeur conventionnelle, dite NULL (à ne pas confondre avec l'initialisation à zéro ou à blanc d'un attribut numérique ou alphabétique !).

> **La contrainte d'entité (ou de relation) est une contrainte d'intégrité imposant que tout attribut qui participe à une clé primaire soit non NULL.**

2.4.4 Concept de vue

Les relations proposées dans la description de la base décrivent la logique de l'ensemble des données que nous voulons traiter.
Cette présentation des données, qui évite toute redondance et assure intégralité et fiabilité, peut être déroutante et difficile à exploiter par un utilisateur non averti.

C'est pourquoi a été créé le concept de VUE.

Une vue permet de définir une structure de données qui pourra être ensuite exploitée comme n'importe quelle relation.
Physiquement, les données ne seront pas stockées dans la base selon cette structure, mais l'utilisateur aura l'impression d'utiliser une table comme les autres.
Une vue peut être créée pour des raisons pratiques (moindre complexité d'utilisation de la base pour un utilisateur non averti), ou aussi pour des raisons de confidentialité (protection de l'accès à certaines données).

3 LE LANGAGES SQL

3.1 INTRODUCTION

Les langages de manipulation de données sont souvent appelés langages de requête car leur utilisation principale est **l'exploitation de données préalablement enregistrées**.

Ainsi, le langage de requête **SQL** [6] propose toutes les fonctions d'interrogation, de mise à jour et de suppression des données (LMD). Il offre aussi les fonctions des trois autres groupes qui sont le LDD, LCD et LCT.
Il s'exprime sous forme de **commandes** interactives ou insérées en programmes. Il assure l'administration des bases de données grâce à un «**dictionnaire**» qui recense les différentes tables, les colonnes de ces tables, les droits d'accès...
~~Certains logiciels sont construits autour des principes SQL (Oracle, Sql-Serveur, MySql) ; d'autres proposent simplement des interfaces avec ce langage de requête (dBase IV, Access).~~
La norme internationale actuelle de SQL est ISO/CEI 9075 :2008 qui apporte quelques fonctions supplémentaires par rapport à la version connue sous le nom SQL3 (SQL-99 ou ISO/CEI9075 :1999).

3.2 LA REQUETE DE BASE DU LANGAGE SQL

La base d'une interrogation SQL est une commande « **SELECT... FROM... WHERE... »** qui permet de récupérer des données de la base.

Un aide-mémoire détaillé est joint en annexe. Nous décrivons succinctement ci-dessous la syntaxe générale.

3.2.1 «SELECT» OU LA PROJECTION DES COLONNES VOULUES

Le mot-clé « **SELECT »** précède l'indication des colonnes désirées dans le résultat final. La projection peut porter sur l'ensemble des colonnes (mot-clé «*») ou sur une liste spécifiée. Quand plusieurs tables sont utilisées, on peut associer le nom de la table au nom de colonne, ce qui permet de lever les ambiguïtés éventuelles (ainsi, CHAUFFEUR.CODE est différencié de VOITURE.CODE).
Pour rejoindre la dimension ensembliste de l'algèbre relationnelle, l'option « **DISTINCT »** permet d'**éliminer les doublons** (attention : elle n'est pas implicite !).
SQL accepte aussi des expressions arithmétiques et offre quelques fonctions statistiques (SUM, AVG...) afin d'effectuer des calculs **au cours** de l'interrogation.

3.2.2 «FROM» OU LE PRODUIT CARTÉSIEN DES TABLES

Le mot-clé « **FROM »** permet de préciser les noms des tables utilisées. Si plus d'une table est mise en œuvre, SQL effectue le **produit cartésien des tables** ; bien que le traitement soit optimisé (avec plus ou moins de bonheur suivant les implémentations...), l'utilisation de plusieurs tables dans une même requête augmente considérablement les consommations de ressources (temps processeur, espace disque).

[6] SQL : Structured Query Language

Notons que SQL permet d'effectuer des produits cartésiens sur une même table afin de réaliser des «auto-jointures» (ou «relations réflexives» selon les termes du modèle entité-relation).

3.2.3 «WHERE» OU LA RESTRICTION AUX LIGNES VOULUES

Le mot-clé « **WHERE »** introduit les différentes restrictions. La condition doit être une expression logique classique appliquée aux colonnes mises en œuvre ou à des expressions de calcul sur ces colonnes.
En plus des opérateurs ordinaires logiques et de comparaison, SQL propose des opérateurs spécifiques dont « IN » qui teste l'appartenance à un ensemble (origine ensembliste oblige).

3.3 EXEMPLE-TYPE DE REQUÊTE SQL SIMPLE

```
SELECT Article.libelle, prix*1.20, en_cours, Stock.mini
    FROM Article, Stock
    WHERE Article.code = Stock.code AND NOT (Article.famille = 'AZR')
        AND prix BETWEEN 100 AND 600 ;
```

Commentaires

1° ligne : projection de colonnes issues de deux tables ; comme il n'y a aucune ambiguïté sur les titres de rubriques, le préfixe indiquant le nom de la table est facultatif.
2° ligne : produit cartésien des deux tables.
3° ligne : restriction de concordance ; un «code» est commun aux deux tables.
4° ligne : restriction utilisateur ; on souhaite tout sauf la famille d'articles «AZR».
5° ligne : l'utilisateur souhaite uniquement les articles dont les prix sont compris entre 100 et 600.
Notez le point–virgule final qui doit terminer obligatoirement toute commande SQL.
La requête exprime donc le besoin : «*liste des articles des familles autres que AZR dont les prix varient entre 100 et 600. On listera les libellés et prix TTC d'articles ainsi que les stock en-cours et mini*». Nous verrons au chapitre 5/ comment passer de la demande utilisateur à la requête SQL.

Question : traduisez le besoin exprimé par la requête SQL :

```
SELECT modele, kilometrage, nom
    FROM Voiture, Chauffeur
    WHERE nom = 'DUPONT' AND type = 'BERLINE';
```

3.4 LES CLAUSES ANNEXES

La requête SQL de base, avec ses clauses SELECT, FROM et WHERE permet d'établir une jointure. SQL propose des extensions intéressantes permettant d'effectuer des **tris** sur la table résultante, des **sous–totaux** et même une **sélection sur ces sous–totaux**.

3.4.1 Clause «GROUP BY»

La clause « **GROUP BY »** met en œuvre un algorithme de détection de ruptures (ce qui entraîne un tri sur le critère de regroupement). Le détail de chaque groupe n'est plus listé ; **seuls les sous–totaux alimentent la table résultante**.

Cette clause est fréquemment utilisée avec les fonctions statistiques SUM, AVG... On effectue parfois deux requêtes successives, l'une générant le détail, l'autre uniquement les sous–totaux.

3.4.2 Clause «HAVING»

La clause « **HAVING »** permet de sélectionner certains groupes établis par la clause GROUP BY, conformément à une condition logique.
La condition est souvent exprimée par rapport aux fonctions statistiques.

Exemple :

```
SELECT famille, COUNT(*), AVG(prix)
    FROM Article GROUP BY famille HAVING COUNT(*) >= 10 ;
```

Il s'agit de compter les articles par familles et de lister les moyennes des prix pour les familles d'au moins 10 articles (tout cela commence à être assez complet...).

3.4.3 Clause «ORDER BY»

La clause « **ORDER BY »** permet simplement de trier les lignes de la table résultante selon un ou plusieurs critères ; cette clause permet à l'utilisateur de préciser un ordre final des lignes.
Notez que le moteur SQL effectue souvent des tris pour des besoins «de service» (optimisation, par exemple).

3.5 REQUÊTES IMBRIQUÉES ET SUCCESSIVES

La requête SQL de base permet d'obtenir de précieux renseignements sur la base de données. Les clauses annexes peuvent en tirer «la substantifique moelle». Il reste pourtant des cas d'interrogations (légitimes) non couverts.
Prenons des exemples. Une commande SELECT FROM WHERE permet de connaître les clients d'un représentant ou les articles par familles. Mais comment retrouver les clients du représentant associé au client MARTIN ou les familles d'articles utilisant des composants que l'on retrouve dans les articles de la famille AZR ?

On atteint ici un niveau de requête sophistiqué... mais bien plus courant en entreprise qu'on ne le pense !
Pour répondre à ces besoins, **SQL permet d'imbriquer des requêtes** : des conditions peuvent porter sur le résultat d'une autre requête (que SQL effectuera au préalable).
Deux opérateurs sont alors utilisables, « **=** » si la requête imbriquée génère une seule ligne, « **IN** » dans le cas contraire.
Le premier cas correspond à une requête de type «*lister les ... dont le ... est égal à celui de ...*», le deuxième cas, à une interrogation de type «*lister les ... dont le ... se retrouve dans ceux de ...*».

Exemple :

```
SELECT libelle, prix
    FROM Article
    WHERE famille =
        (SELECT famille
            FROM Article
            WHERE code = 123);
```

«Liste des libellés et prix des articles de la même famille que l'article de code 123».

Exemple :

```
SELECT libelle, prix
    FROM Article
    WHERE code IN
        (SELECT DISTINCT Stock.code
            FROM Stock);
```

«Liste des libellés et prix des articles ayant subi au moins un mouvement de stocks (et donc présents dans la table STOCK)».

Encore plus fort : **toutes les requêtes imbriquées peuvent s'exprimer à l'aide de requêtes successives** à condition de lancer d'abord la requête imbriquée et de mémoriser son résultat dans une table (option INTO dans Access).

Ainsi l'exemple précédent peut s'exprimer :

```
SELECT DISTINCT code INTO ArtMouv
    FROM Stock;
```

« Liste des articles ayant subi des mouvements de stock »

```
SELECT libelle, prix
    FROM Article, ArtMouv
    WHERE Article.code = ArtMouv.code;
```

« Liste des articles existant à la fois dans TARIF et ARTMOUV »; c'est bien le résultat voulu.

Toujours plus fort : toutes les requêtes imbriquées peuvent s'exprimer à l'aide de produits cartésiens.

Ainsi le premier exemple peut s'exprimer :

```
SELECT Y.libelle, Y.prix
    FROM Article AS X, Article AS Y,
    WHERE X.famille = Y.famille
        AND X.code = 123
        AND Y.code <> 123 ;
```

ou bien pour la dernière ligne **AND** Y.code <> X.code ;

Commentaires : on est en présence d'une auto–jointure (la table Tarif est jointe avec elle-même), d'où les synonymes «X» et «Y» donnés aux tables ; la première table, X, est considérée comme la table de référence, celle où l'on va chercher l'article de code 123 ; la deuxième table, Y, est celle où l'on va puiser les articles de même famille que l'article de référence, à l'exception toutefois de l'article de code 123 (qui est forcément de la même famille que lui-même !).

Effectivement, tout cela se complique... Il est important de sentir que plusieurs solutions sont possibles pour atteindre un même résultat.

En règle générale, les requêtes imbriquées sont plus «naturelles» que les deux équivalents ci-dessus car elles correspondent mieux à l'expression du besoin.

3.6 CONSTRUCTION D'UNE REQUÊTE SQL

LA REQUÊTE SQL

Instruction permettant d'extraire des données à partir d'une base.

Comme toute instruction d'un langage quelconque, une instruction SQL a une grammaire, une syntaxe et un vocabulaire propre.

Cette requête permettra de "manipuler" les informations (données) contenues dans la base et, si tout va bien (!), d'obtenir un résultat qui devrait pouvoir satisfaire le besoin de l'utilisateur.

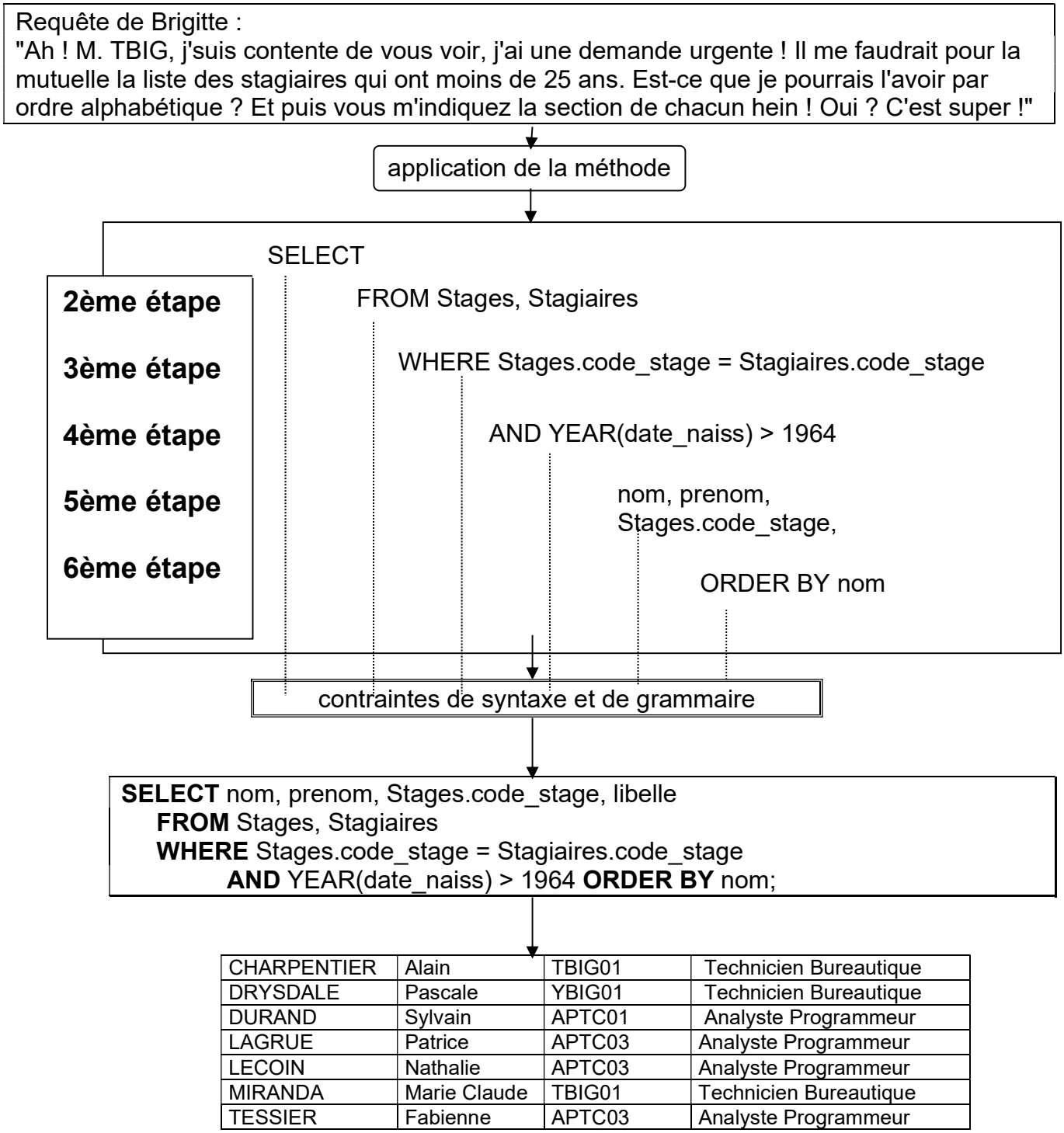
ET LES SIX ÉTAPES DE LA MÉTHODE PROPOSEE

Les six étapes de la méthode proposée permettent d'arriver à une requête d'interrogation utilisable sur une base de données.

L'instruction générique d'extraction en SQL : **SELECT**

1ère étape	énumération, recensement	Prépare les étapes 2 à 6. Affine et restreint la requête si nécessaire par les clauses **DISTINCT** ou **GROUP BY.**
2ème étape	exécution du produit cartésien	Clause **FROM.**
3ème étape	restriction de concordance	Clause **WHERE.** Elle est à utiliser uniquement en cas de base multi–tables (étape 2).
4ème étape	restriction de condition utilisateur	Clause **WHERE.** Elle sera associée par un opérateur logique AND si l'étape 3 a déjà imposé l'utilisation d'un WHERE.
5ème étape	projection sur les colonnes	Clause **implicite** suivant immédiatement le SELECT. Attention à la qualification des colonnes homonymes en cas de base multi–tables
6ème étape	Séquence de présentation	Clause **ORDER BY.**

Exemple:



CHARPENTIER	Alain	TBIG01	Technicien Bureautique
DRYSDALE	Pascale	YBIG01	Technicien Bureautique
DURAND	Sylvain	APTC01	Analyste Programmeur
LAGRUE	Patrice	APTC03	Analyste Programmeur
LECOIN	Nathalie	APTC03	Analyste Programmeur
MIRANDA	Marie Claude	TBIG01	Technicien Bureautique
TESSIER	Fabienne	APTC03	Analyste Programmeur

Image du résultat obtenu et répondant à la demande initiale de Brigitte.

3.7 LES ORDRES DE CREATION ET MODIFICATION

La définition des données permet la description de tous les objets manipulés par le SGBD (tables, synonymes, vues).

3.7.1 Le verbe CREATE

Cet ordre permet la création de nouveaux objets dans la base : tables, vues, synonymes.

Syntaxe : **CREATE TABLE** NomTable (champ type (taille) [**NOT NULL**] ,**CONSTRAINT**…

L'exemple ci-dessous crée une nouvelle table nommée *CetteTable* comportant deux champs texte.

CREATE TABLE *CetteTable (prenom TEXT, nom TEXT);*

L'exemple ci-dessous crée une nouvelle table nommée *MaTable* comportant deux champs texte, un champ Date/heure et un index unique composé des trois champs

CREATE TABLE *MaTable* ***(****prenom TEXT, nom TEXT, [date De Naissance] DATETIME,* ***CONSTRAINT*** *NomConstraint* ***UNIQUE (****prenom, nom, [date De Naissance]****));***

L'exemple ci-dessous crée une nouvelle table comportant deux champs texte et un champ de nombre entier. Le champ *SSN* constitue la clé primaire.

CREATE TABLE *NouvelleTable* ***(****prenom TEXT, nom TEXT, SSN INTEGER,* ***CONSTRAINT*** *NomConstraint* ***PRIMARY KEY (****SSN****));***

3.7.2 Le verbe INSERT

Ajoute un ou plusieurs enregistrements à une table. C'est ce qu'on appelle une requête Ajout.

Syntaxe: **INSERT INTO** target [(field1[, field2[, ...]])] **VALUES** (value1[, value2[, ...])

L'exemple ci-dessous présente la sélection de tous les enregistrements de la table *NouveauxClients* et les ajoute à la table Clients. Si les colonnes individuelles ne sont pas désignées, les noms de colonne de la table SELECT doivent correspondre exactement à ceux de la table INSERT INTO.

INSERT INTO *Clients* ***SELECT*** *** ***FROM*** *NouveauxClients ;*

L'exemple ci-dessous présente la création d'un nouvel enregistrement dans la table Employés :

INSERT INTO *Employés (Prénom, Nom, Fonction)* ***VALUES*** *('Dupont', 'Serge', 'Stagiaire') ;*

L'exemple ci-dessous suivant présente la sélection, à partir d'une table Stagiaires existante, de tous les stagiaires embauchés depuis plus de 30 jours, et ajoute leurs enregistrements à la table Employés.

INSERT INTO *Employés* ***SELECT*** *Stagiaires.** ***FROM*** *Stagiaires* ***WHERE*** *dateEmbauche < now() - 30 ;*

3.7.3 Le verbe UPDATE

Crée une requête de mise à jour qui modifie les valeurs des champs d'une table spécifiée, selon des critères déterminés.

Syntaxe: **UPDATE** table **SET** valeur **WHERE** critère ;

UPDATE est particulièrement utile lorsque vous désirez modifier simultanément de nombreux enregistrements ou que les enregistrements que vous souhaitez modifier résident dans différentes tables.

Vous pouvez modifier simultanément plusieurs champs. Dans l'exemple suivant, les valeurs de *MontantCommande* sont augmentées de 10 pour cent tandis que les valeurs de Port augmentent de 3 pour cent pour les transporteurs résidant au Royaume Uni.

***UPDATE** Commandes **SET** [montantCommande] = [montantCommande] * 1.1, port = port * 1.03 **WHERE** [paysLivraison] = 'RU';*

3.7.4 Le verbe DELETE

Crée une requête Suppression qui supprime des enregistrements dans une ou dans plusieurs des tables mentionnées dans la clause FROM qui correspond à la clause WHERE.

Syntaxe: **DELETE** [table.*] **FROM** table **WHERE** critère

DELETE est particulièrement utile lorsque vous désirez supprimer de nombreux enregistrements en même temps.

Supprime les enregistrements d'employé ayant la fonction Stagiaire.

***DELETE** * **FROM** Employés **WHERE** fonction = 'Stagiaire;*

3.7.5 L'instruction INNER JOIN

Fusionne les enregistrements de deux tables lorsqu'un champ commun contient des valeurs identiques.

Syntaxe: **FROM** table1 **INNER JOIN** table2 **ON** table1.field1 opérateur table2.field2

L'exemple qui suit montre comment réaliser une jointure entre les tables Catégories et Produits, sur la base du champ Nom de catégorie :

***SELECT** [Nom de catégorie], [Nom du produit] **FROM** Catégories **INNER JOIN** Produits*
***ON** Catégories.[Nom de catégorie] = Produits.[Nom de catégorie];*

3.7.6 L'instruction UNION

Crée une requête Union, qui fusionne les résultats de deux, ou plusieurs, requêtes ou tables indépendantes.

Syntaxe: [**TABLE**] Requête1 **UNION** [**ALL**] [**TABLE**] ….requêteN [...]]

Requête1 à requêteN correspondent aux requêtes SELECT, nom d'une requête enregistrée ou d'une table enregistrée, précédé du mot clé TABLE.

Remarques: Vous pouvez fusionner les résultats de deux, ou plusieurs, requêtes, tables ou instructions SELECT, dans n'importe quel ordre, en une unique opération UNION.

Dans l'exemple suivant, on fusionne une table existante (Nouveaux Comptes) avec une instruction SELECT.

***TABLE** [Nouveaux Comptes] **UNION ALL SELECT** * **FROM** Clients*
***WHERE** [montant Commande] > 1000;*

Par défaut, l'opération UNION ne renvoie aucun enregistrement en double mais vous pouvez ajouter le prédicat ALL pour obtenir de façon certaine que tous les enregistrements soient renvoyés. La requête s'exécutera plus rapidement par ailleurs.