Dr **KONAN HYACINTHE**

Support de cours
Base de données

Pré requis

- Notion de fichier
- Notion de variables

Objectifs généraux

A la fin de ce module, l'étudiant doit être capable de :

- Découvrir les **concepts** de base des bases de données.
- Analyser une **étude de cas** donné afin de dégager le **modèle entités/associations** et le **modèle relationnel** associé.
- Apprendre à utiliser un langage normalisé d'accès aux données (**SQL**).

SOMMAIRE

Chapitre 1 : Introduction aux bases de données

Objectifs spécifiques

A la fin de ce chapitre, l'étudiant doit être capable de :

- **Définir** une base de données.
- **Expliquer** l'intérêt de l'approche **base de données** par rapport à celle à **fichiers**.
- **Définir** un Système de Gestion de Bases de Données (**SGBD**)
- **Enumérer** les fonctions d'un SGBD
- **Différencier** entre les niveaux d'abstraction lors de la conception d'une base de données.

Plan du chapitre

I. Introduction
II. Les systèmes à fichiers
III. Les bases de données
IV. Les SGBD
V. Les niveaux d'abstraction

I. Introduction

Toute entreprise (établissements d'enseignement, ministères, banques, agences de voyages, transport, santé, etc.) dispose d'un ensemble de données qu'elle propose de **stocker**, **organiser**, **manipuler** et **exploiter**.

- **Stockage et organisation :** saisir, insérer les informations et les enregistrer dans les emplacements appropriés dans le système.
- **Manipulation :** chercher, sélectionner, mettre à jour et supprimer les données archivées dans le système.
- **Exploitation :** récupérer les données et les informations nécessaires afin de prendre une décision.

Au fil des années, les **quantités** de données deviennent **de plus en plus grandes**. Certains experts (dans le domaine des statistiques) estiment même que le volume de données collectées par une entreprise double tous les vingt mois.

Explosion vertigineuse du volume de données au sein des entreprises.
Recourir à un **moyen efficace** de stockage, de manipulation et d'exploitation de données.

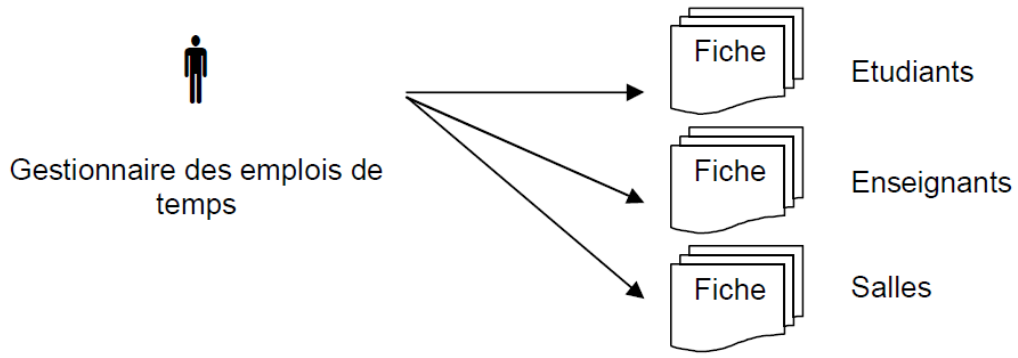
II. Les systèmes à fichiers

II.1. Définitions

Fichier :
Un fichier est un **récipient** de données identifié par un nom constitué par un **ensemble de fiches** contenant des informations système ou utilisateur. Les informations relatives à un même sujet sont décrites sur une fiche.

Gestionnaire de fichiers :
structuré autour d'un noyau appelé analyseur, le **Gestionnaire de fichiers** assure les fonctions de base, à savoir la création/destruction des fichiers, lecture/écriture d'une suite d'octets à un certain emplacement dans un fichier, l'allocation de la mémoire, la localisation et la recherche des fichiers sur les volumes mémoires.

Exemples :



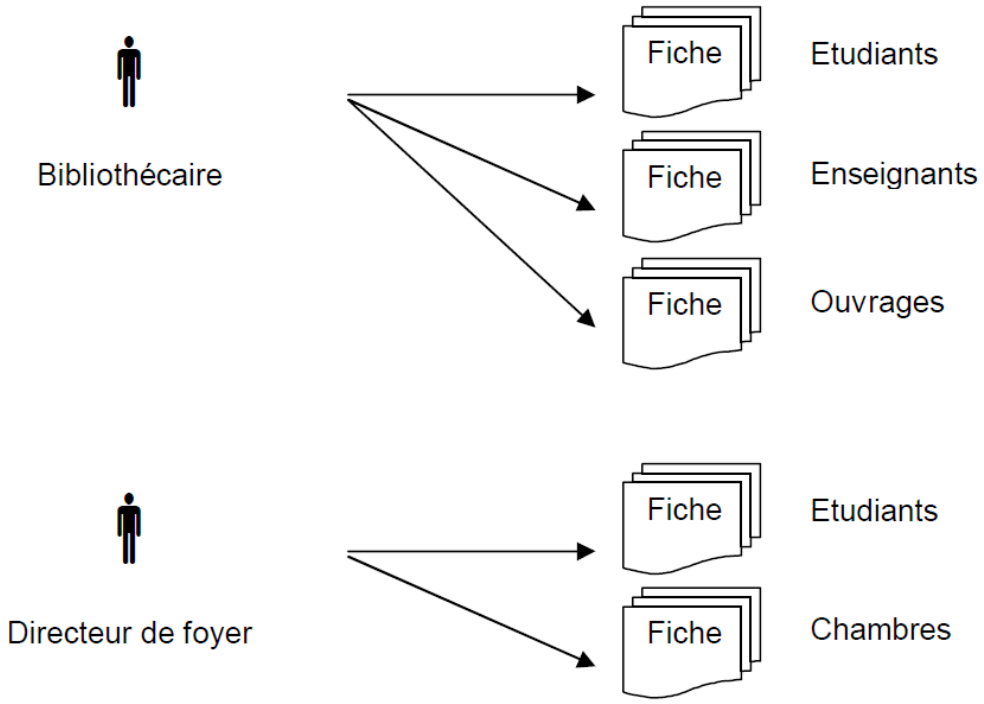


Figure 1 : Organisation de données en système à fichier

II.2. Limites

Les **systèmes à fichiers** soulèvent certaines limites à savoir :

- **Dispersion** des informations.
- **Contrôle différé** de données.
- Risque **d'incohérence** de données.
- **Redondance** de données.
- **Difficulté** d'accès, d'exploitation, d'organisation et de maintenance.

III. Les bases de données

III.1. Définition

Une **BASE DE DONNEES** est un ensemble de données d'entreprise enregistrées sur des **supports** accessibles **par l'ordinateur** pour satisfaire **simultanément** plusieurs utilisateurs de façon sélective et en **temps opportun**.

Donc, une base de données nécessite :

- Un **espace de stockage**.
- Une **structuration** et **relations** entre les données (sémantique).
- Un **logiciel** permettant **l'accès** aux données stockées pour la **recherche** et la **mise à jour** de l'information.

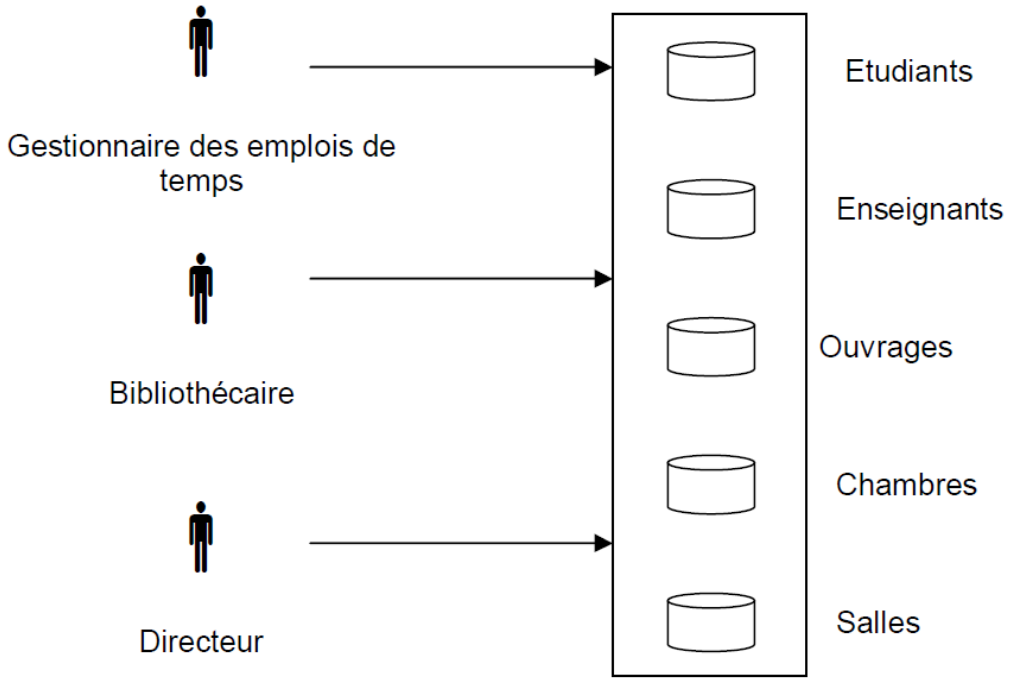


Figure 2 : Organisation de données en système à base de données

III.2. Types d'utilisateurs de bases de données

On distingue essentiellement trois types différents d'utilisateurs bases de données à savoir :

- **Administrateurs** de bases de données : Ils gèrent la base (les **accès**, les **droits** des utilisateurs, les **sauvegardes**, les **restaurations**)
- **Concepteurs** de bases de données et développeurs d'application : Ils représentent ceux qui **définissent**, **décrivent** et **créent** la base de données.
- **Utilisateurs** finaux : manipulent la base de données. Il est possible de distinguer des **familles** utilisateurs avec des droits différents vis-à-vis de l'accès à la base. On suppose qu'ils n'ont aucune connaissance sur les bases de données.

IV. Les Système de Gestion de Base de Données (SGBD)

IV.1. Définition

Un **SGBD** (Système de Gestion de Bases de Données) est un logiciel qui permet à des utilisateurs de définir, créer, mettre à jour une **base de données** et d'en contrôler l'accès. C'est un outil qui agit comme interface entre la **base de données** et **l'utilisateur**. Il fournit les moyens pour **définir**, **contrôler**, **mémoriser**, **manipuler** et **traiter** les données tout en assurant la **sécurité**, **l'intégrité** et la **confidentialité** indispensables dans un environnement multi utilisateurs.

IV.2. Objectifs

On distingue essentiellement trois fonctions des SGBD à savoir :

- **Description** des données : Langage de Définition de Données (**LDD**).
- **Recherche**, **manipulation** et **mise à jour** de données : Langage de Manipulation de Données (**LMD**).
- **Contrôle** de **l'intégrité** et **sécurité** de données : Langage de Contrôle de Données (**LCD**).

En d'autres termes, les objectifs d'un SGBD peuvent être résumés dans les points suivants :

- **Intégrité** de données.
- **Indépendance** données / programmes : Ajout d'un nouveau champ ne provoque aucun problème vis à vis des autres champs.
- **Manipulation facile** de données : un utilisateur **non informaticien** peut manipuler simplement les données.
- **Sécurité** et administration de données.
- **Efficacité** d'accès aux données.
- **Redondance** contrôlée de données.
- **Partage** de données.

IV.3. Historique

Jusqu'aux années 60 : Organisation classique en fichiers : systèmes à fichiers.
Fin des années 60 : Première génération : apparition des premiers SGBD.

- Séparation de la description des données de leur manipulation par les programmes d'application.
- Basés sur des **modèles navigationnels** :
- **Modèles hiérarchiques** **:** modèles père fils, **structures d'arbres**, (Exemples :
    - **Adabas**, ou **IMS** (Information Management System))
    - **Modèles réseaux** **:** modèle père fils, **structure de graphes**. (Exemples : **SOCRATE**, **IIDS**, **IDMS**, **IDS2**, **IMS2**).

A partir de 1970 : Deuxième génération de SGBD à partir du **Modèle relationnel** **(SGBDR)**.

- Enrichissement et simplification des SGBD afin de faciliter l'accès aux données pour les utilisateurs.
- Modèle mathématique de base : **l'algèbre relationnelle**
- Exemples : **ORACLE**, **SQL Server**, **DB2**, **Access**, **PostgreSQL**, **MySQL**.

Début des années 80 : Troisième génération de SGBD basées sur des modèles plus complexes :

- **SGBD déductifs :** modèle logique, algèbre booléenne, souvent avec un autre SGBD (généralement relationnel) qui gère le stockage, présence d'un **moteur d'inférence**.
- **SGBD à objets** (SGBDOO) : modèles inspirés des langages de programmation orientée objet tels que Java, C++ utilisation de l'encapsulation, l'héritage, le polymorphisme,
- Exemples : **ONTOS**, **VERSANT**, **ORION**.

V. Les niveaux d'abstraction

La **conception** d'une base de données passe essentiellement, comme le montre la **figure 3**, par trois niveaux d'abstraction à savoir :

V.1. Niveau externe

Ce niveau présente une **vue de l'organisation** (ou d'une partie) par des **utilisateurs** ou des applications. En effet, il prend en charge le problème du **dialogue avec les utilisateurs**, c'est-à-dire l'analyse des demandes de l'utilisateur, le contrôle des droits d'accès de l'utilisateur, la présentation des résultats.

V.2. Niveau conceptuel

Il s'agit d'une **représentation abstraite** globale de l'organisation et de ses mécanismes de gestion. Ce niveau assure les fonctions de contrôle global (optimisation globale des requêtes, gestion des conflits d'accès simultanés, contrôle général de la cohérence de l'ensemble etc.).

V.3. Niveau interne

Ce niveau s'occupe du **stockage** des données dans les **supports physiques** et de la gestion des structures de mémorisation et d'accès (gestion des index, des clés, ...)

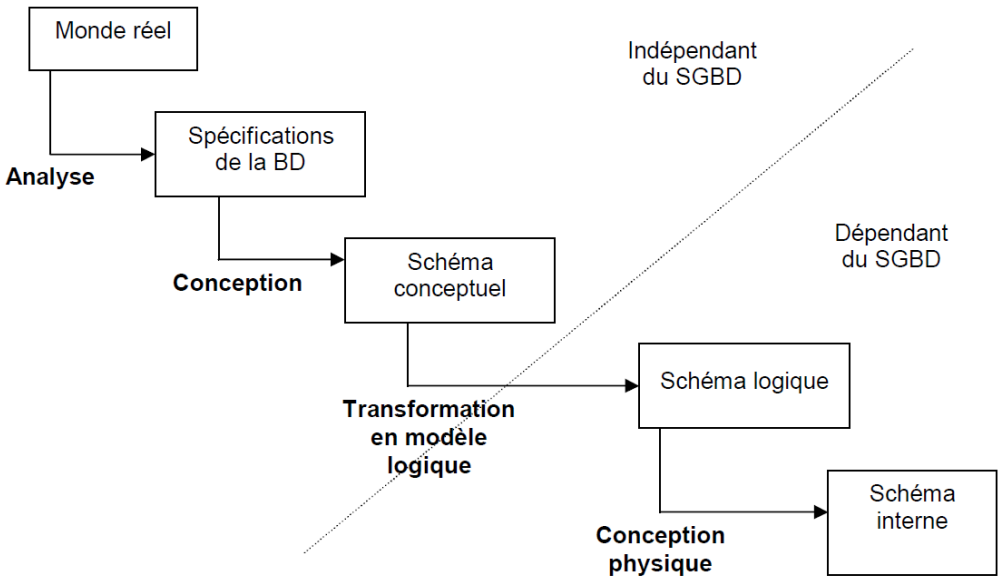


Figure 3 : Niveaux d'abstraction

Chapitre 2 : Le modèle Entités/Associations

Objectifs spécifiques

A la fin de ce chapitre, l'étudiant doit être capable de :

- **Assimiler** la sémantique du modèle **Entités/Associations**
- **Utiliser** le formalisme du modèle Entités/Associations
- **Distinguer** les différents types d'attributs
- **Analyser** une **étude de cas** donnée
- **Modéliser** en **Entités/Associations**

Plan du chapitre

I. Généralités
II. Concepts de base
III. Associations et cardinalités
IV. Démarche à suivre pour produire un schéma E/A.

I. Généralités

- Le modèle Entités/Associations est généralement connu.
- C'est un modèle conceptuel conçu dans les années 1970 qui résulte des travaux de BACHMAN, CHEN, TARDIEU.
- Il est essentiellement utilisé pour la phase de **conception initiale**.
- Il utilise une **représentation graphique**.
- Mise en œuvre de la base de données : transformation du schéma E/A en un schéma logique de SGBD.

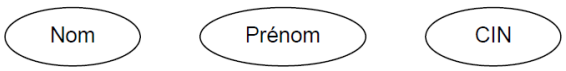
II. Concepts de base

II.1. Attribut

Définition : Un **attribut** est défini comme étant le champ ou **la plus petite unité** de données possédant un nom.

Exemples : Nom, prénom, date_naissance, immatricule_voiture, raison sociale, etc.

Notation : On présente les attributs par des **ellipses** contenant leurs noms.
Exemples :



Propriétés :

- ***Attribut simple* :** attribut **non décomposable** en d'autres attributs.
  Exemples : Nom, Prénom, NCI, Email, Téléphone.
- ***Attribut composé* :** la valeur de l'attribut est une **concaténation** des valeurs de plusieurs attributs simples.
  Exemple : Adresse (est la concaténation des attributs : Rue, Code_postal et Ville).
- ***Attribut dérivé* :** la valeur de l'attribut est **calculée** ou **déduite** à partir des valeurs des autres attributs.
  Exemples : Age, Moyenne, Durée
- ***Valeur nulle :*** pour un attribut, c'est une valeur non définie.
  Exemple : Pour un client dont on ne connaît pas sa date de naissance, l'attribut **date_naissance** prend la valeur **NULL**.

Type d'attribut : Entier, Réel, Date, Chaîne de caractères

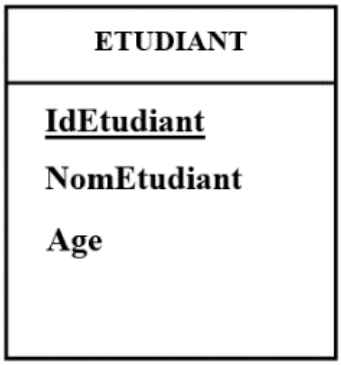
Domaine d'attribut : Ensemble de valeurs admissibles pour un ou plusieurs attributs.
Exemple : Si le prix des produits est compris entre **1**F et **15**F, alors le domaine de l'attribut **prix** est **{1, …, 15}**.

II.2. Entité

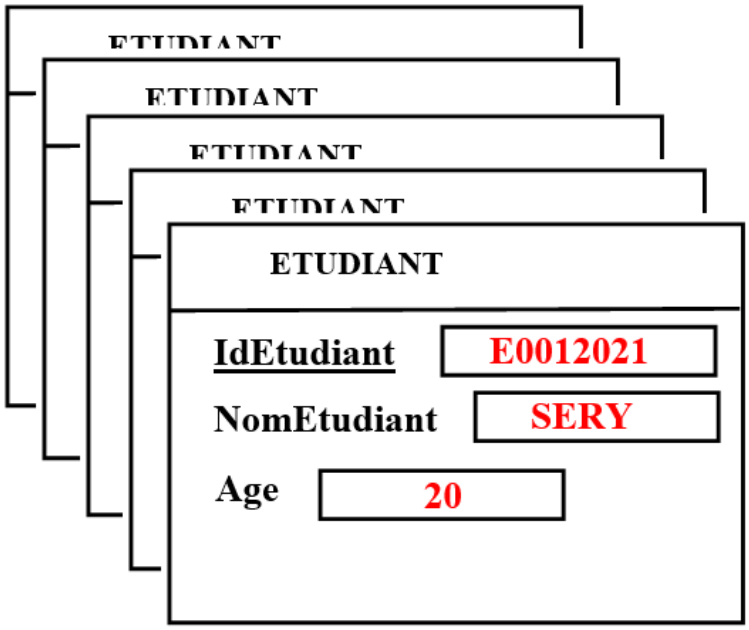
Définitions :

- Une **entité** permet de définir de façon conceptuelle une **catégorie d'objet** de l'univers du discours = (sujet, thème) dont tous les membres partagent les mêmes caractéristiques.
- Une **occurrence d'entité** est constituée par l'ensemble des valeurs de chacune des propriétés d'une d'entité.

L'entité ETUDIANT



5 **occurrences** de l'entité ETUDIANT



Remarque : Les bases de données sont basées sur la **théorie des ensembles**. **L'entité** est **l'ensemble**. Les **occurrences** en sont les **éléments**.

Identifiant d'une entité : caractérise de façon unique les occurrences d'une d'entité.

Exemple : L'attribut **CNI** de l'entité **Personne** : Toute personne **a un seul** N° de carte d'identité nationale qui le distingue des autres.

Notation : Chaque entité est représentée par un **rectangle** et doit avoir un **identifiant** qui doit être **souligné**.

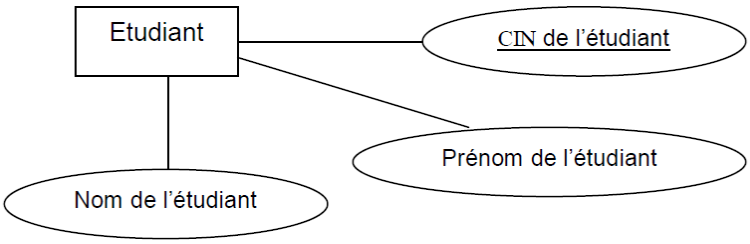
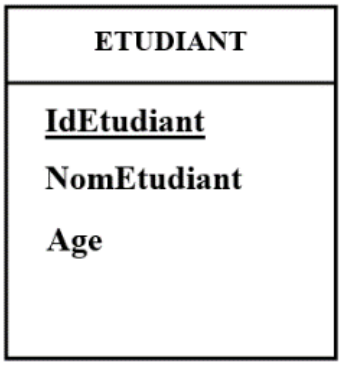


Figure 4 : Exemple d'entité avec ses attributs

L'entité ETUDIANT

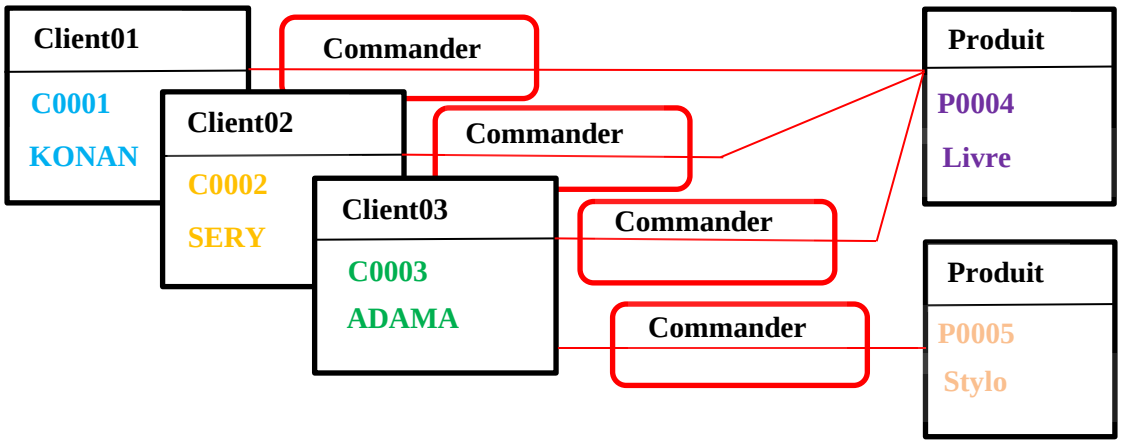


III. Associations et Cardinalités

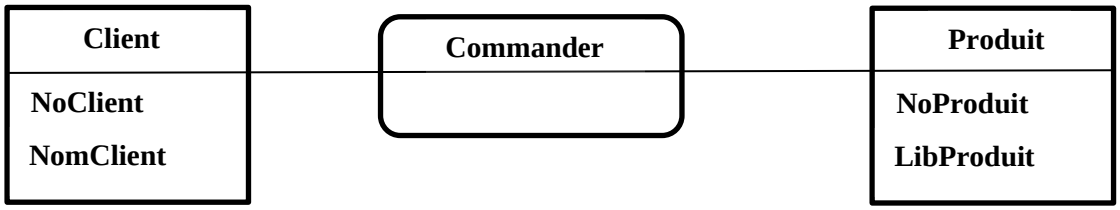
III.1. Associations

Définition : Une **ASSOCIATION** est une liaison perçue entre plusieurs entités. Elle présente un lien où chaque entité liée joue un rôle bien déterminé.

Exemple : Les clients commandent des produits.
En termes d'occurrence on a :



Que nous modélisons par :



Ou par :

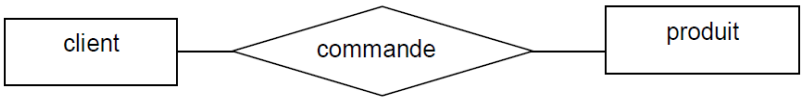


Figure 5 : Exemple d'association entre deux entités

On dit qu'il **existe des occurrences** d'entités **client** et **produit** qui **participent** à la l'association **commande**.

III.2. Cardinalités

Définition : Les associations sont caractérisées par des **CARDINALITES**. La cardinalité, notée **(Min, Max)** attachée à une entité indique les nombres **minimal (Min)** et **maximal (Max)** **d'occurrences** d'associations pour une occurrence de cette entité.

Remarque : Une cardinalité se lit dans le sens **entité** vers **association**.

III.3. Types de cardinalités

Notations :
Min = **0** (**une** occurrence **peut** exister sans participer à l'association)
Min = **1** (**Toutes** les occurrences participent à l'association)
Max = **1** (Les occurrences qui participent à l'association ne le font **qu'une seule fois**)
Max = **N** (Les occurrences qui participent à l'association peuvent le faire **plusieurs fois**)

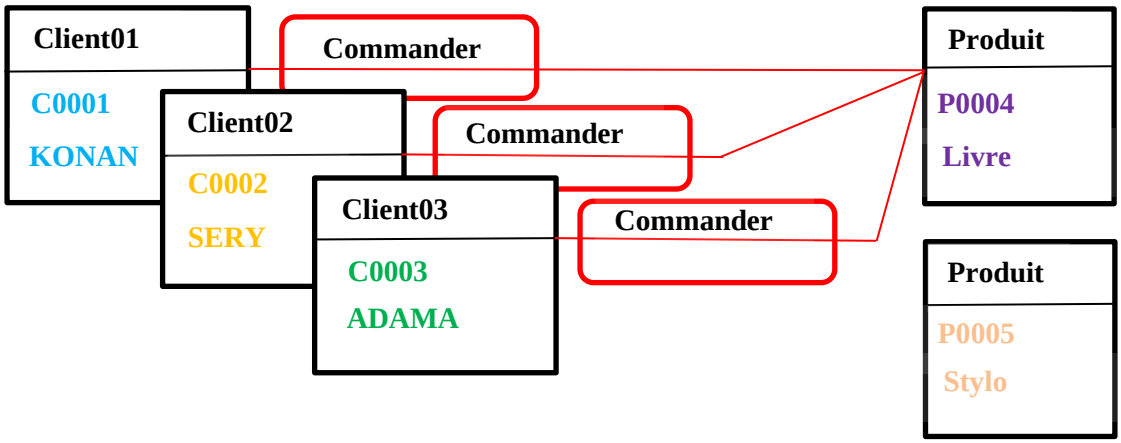
Définition : Règle de gestion Une règle de gestion est un énoncé qui spécifie le fonctionnement de l'entreprise. Elle permet de définir les **cardinalités**.

Exemples :

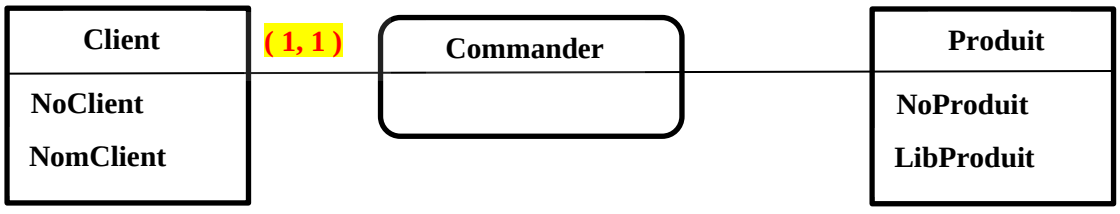
- Cardinalité (1, 1) :
Règle de gestion :

- **TOUS** les clients commandent un produit : **Min = 1**.
- **IL N'EXISTE PAS** de client qui commande **PLUS DE UN** produit : **Max = 1**. On obtient la cardinalité **(**1**,** 1**)** coté client.

En termes d'occurrence on a :



Que nous modélisons par **(1, 1)** :



Ou :

Figure 6 : Exemple d'association de type **(1, 1)**

- Cardinalité (1, N) :

Règle de gestion coté client :

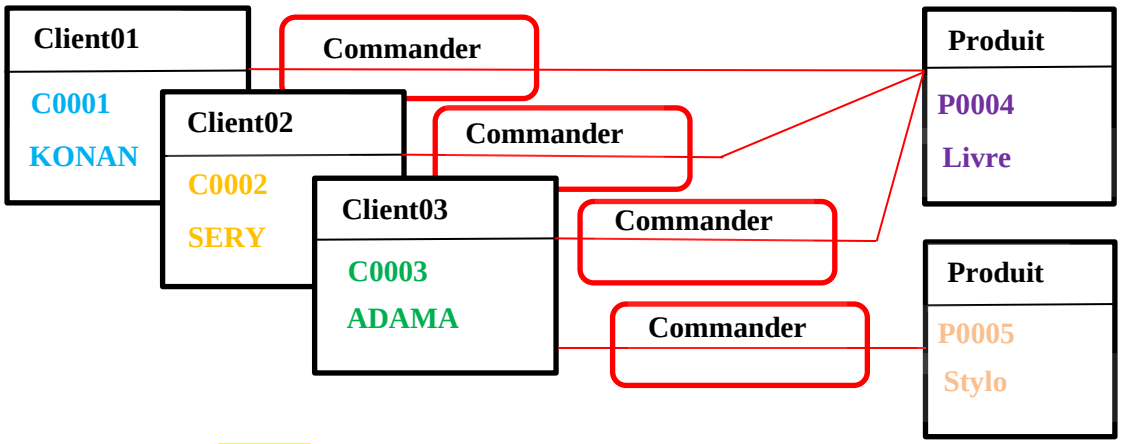
- **TOUS** les clients commandent un produit : **Min = 1**.
- **IL EXISTE** un client qui commande **PLUS DE UN** produit : **Max = N**. On obtient la cardinalité **(1, N)** coté client.

Figure 7 : Exemple d'association de type **(1, N)**

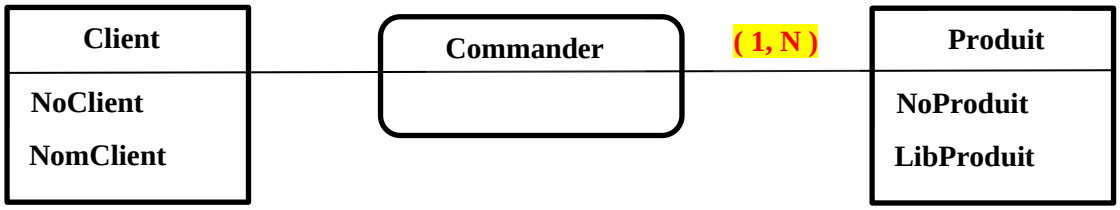
Remarque : S'IL EXISTE des clients qui ne commandent pas de produit **ALORS Min = 1**.

Règle de gestion coté produit :

- **TOUS** les **produits** sont commandés par un **client** : **Min = 1**.
- **IL EXISTE** un **produit** qui est commandé par **PLUS DE UN client** : **Max = N**. On obtient la cardinalité **(1, N)** coté **produit**.



Que nous modélisons par **(1, N)** coté Produit :



Ou :

III.4. Type d'association

C'est le couple **(Max1, Max2)** formé des cardinalités maximale de chaque coté des entités qui participe à l'association.

- Association de type (M, N) :
Règle de gestion : Un client commande nécessairement un produit. Il peut exister un client qui en commandeplusieurs. Un produit est nécessairement commandé par un client. Il peut être commandé plusieurs fois. Dans ce cas l'association COMMANDE est de type **(M, N)**.

Figure 8 : Exemple d'association de type **(M-N)**

III.5. Attributs d'association

Dans une association de type **M-N**, il est possible de caractériser l'association par des attributs.

Exemple :
Règle de gestion : Une commande est passée à une date donnée et concerne une quantité de produit fixe.

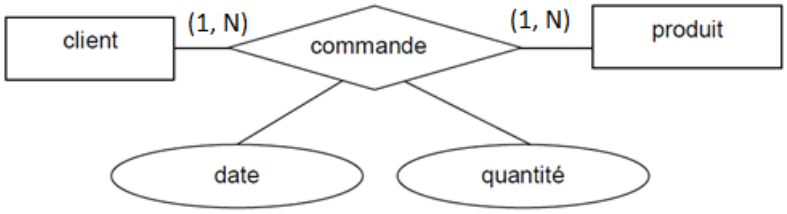


Figure 9 : Exemple d'association de type **M-N**

Remarque :
- On peut avoir une association réflexive au niveau de la même entité.
- Une association peut l'être entre plus de deux entités.

IV. Démarche à suivre pour produire un schéma E/A

IV.1. Démarche

Afin de pouvoir produire un schéma Entité / Association relatif aux spécifications d'une **étude de cas**, on procède comme suit :

1. Recueil des besoins et identification des différents **attributs**.
2. Regrouper les attributs par **entités**.
3. Identifier les **associations** entre les **entités** ainsi que les **attributs** y associés.
4. Evaluer les **cardinalités** des associations.

IV.2. Exemple illustratif : Gestion simplifié de stock

Spécifications :

Les clients sont caractérisés par un numéro de client, un nom, un prénom, une date de naissance et une adresse postale (rue, code postal et ville). Ils commandent une quantité donnée des produits à une date donnée. Les produits sont caractérisés par un numéro de produit, une désignation et un prix unitaire. Chaque produit est fourni par un fournisseur unique (mais un fournisseur peut fournir plusieurs produits). Les fournisseurs sont caractérisés par un numéro de fournisseur, une raison sociale, une adresse email et une adresse postale.

Spécifications :

Les **CLIENTs** sont caractérisés par un numéro (**NumCl**) de client, un nom (**NomCl**), un prénom(**PrenomCl**), une date de naissance(**DateNaisCl**) et une adresse postale (**AdrCl**) (rue, code postal et ville). Ils **COMMANDE**nt une quantité (**QtePc**) donnée des **PRODUIT**s à une date donnée (**DateCd**). Les produits sont caractérisés par un numéro de produit (**NumPd**), une désignation (**designPd**) et un prix unitaire (**PuPd**). Chaque produit est fourni par un fournisseur unique (mais un fournisseur peut fournir plusieurs produits). Les fournisseurs sont caractérisés par un numéro de fournisseur (**NumFr**), une raison sociale (**RsFr**), une adresse email (**EmailFr**) et une adresse postale (**AdrFr**).

Solution :

Les différents attributs associés à ces spécifications peuvent être résumés comme suit :

Nom de l'attribut	Désignation de l'attribut
NumCl	Numéro du client
NomCl	Nom du client
prenomCl	Prénom du client
datenaisCl	Date de naissance du client
AdrCl	Adresse du client
QtePc	Quantité de produits commandés
DateCd	Date de la commande
NumPd	Numéro du produit
designPd	Désignation du produit
PuPd	Prix unitaire du produit
NumFr	Numéro du fournisseur
RsFr	raison sociale du fournisseur
EmailFr	Adresse email du fournisseur
AdrFr	Adresse du fournisseur

Les entités avec leurs attributs sont :

Nom de l'entité	Attribut de l'entité
CLIENT	numCl, nomCl, prenomCl, datenaisCl, adrCl
PRODUIT	numPd, designPd, puPd
FOURNISSEUR	numFr, rasFr, emailFr, adrFr

Les associations entre les entités :

Nom de l'association	Entités participantes	Attribut associés
COMMANDER	**CLIENT** et **PRODUIT**	qtePc, dateCd
FOURNIR	**FOURNISSEUR** et **PRODUIT**	

Enfin, le **modèle E/A** se présente comme suit :

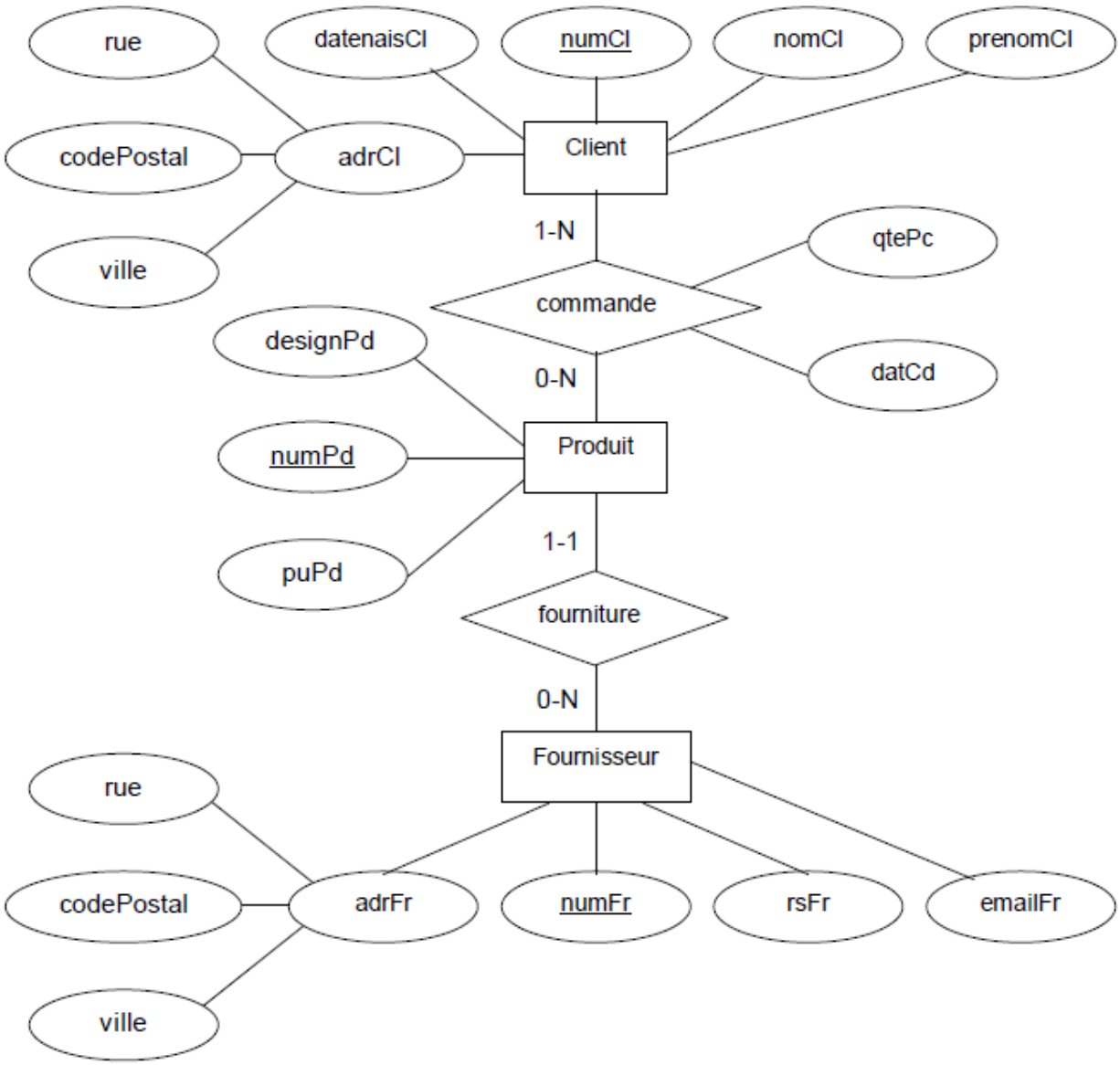
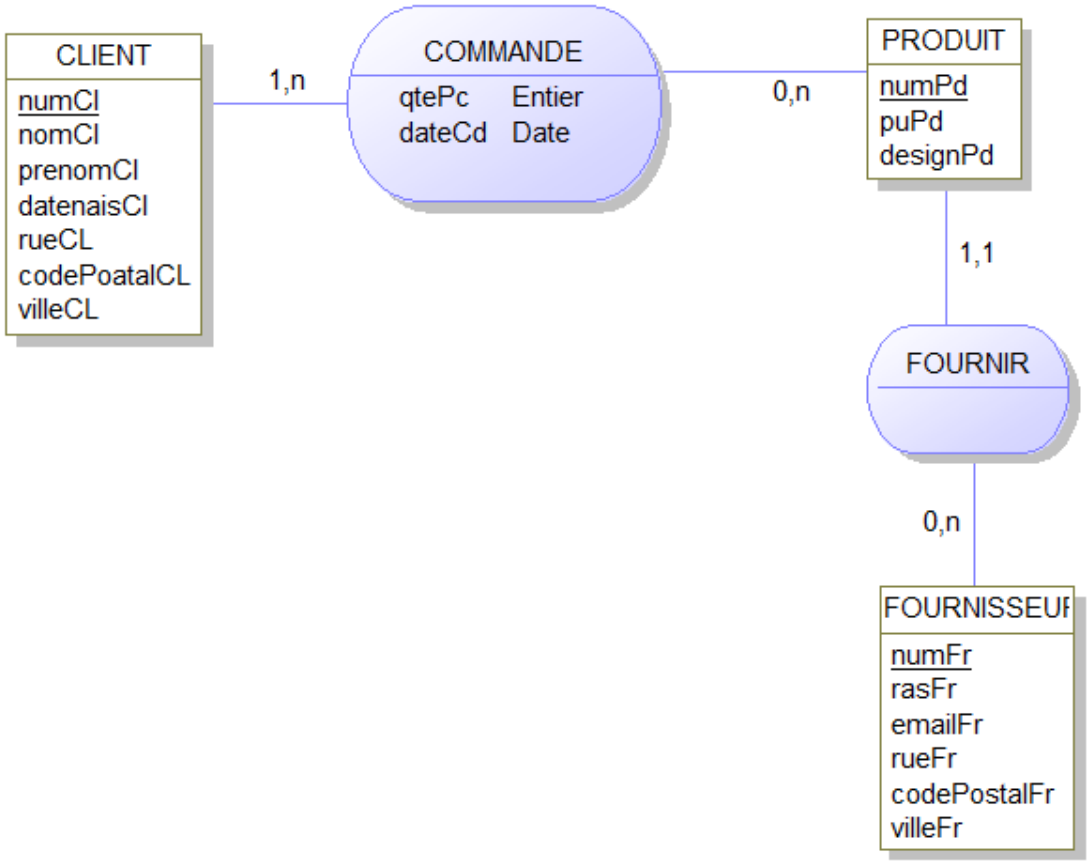


Figure 10 : Modèle E/A de l'exemple illustratif

1 Lancer PowerAMC
2
3 Aller dans Outils/Option du modèle
4 Dans notation, choisir **E/R + Merise**



Chapitre 3 : Le modèle relationnel

Objectifs spécifiques

A la fin de ce chapitre, l'étudiant doit être capable de :

- Apprendre les notions de base du modèle relationnel
- Identifier les correspondances avec le modèle E/A
- Traduire un **modèle E/A** en un **modèle relationnel**
- Dégager les dépendances fonctionnelles
- Normaliser une relation

Plan du chapitre

I. Généralités
II. Concepts de base
III. Traduction E/A - relationnel
IV. Les dépendances fonctionnelles
V. Normalisation

I. Généralités

Bien que tous les outils permettent de modéliser un schéma logique relationnel, aucun ne peut offrir un processus de **normalisation automatique**. Il est donc nécessaire de maîtriser à la fois le modèle de données et les **principes de normalisation** afin de concevoir des bases de données **cohérentes**.

Terminologie
Une **RELATION** possède des **ATTRIBUTS**. Chaque attribut prend sa valeur dans un **DOMAINE** (ensemble de valeurs). Chaque élément d'une relation est appelé **UPLET** ou **N-UPLET** (n désignant le nombre d'attributs de la relation et étant aussi appelé degré de la relation). L'ensemble des n-uplets désigne la relation en **EXTENSION**.

Notations
R (A1, ..., An) désigne la relation **R** composée de **n** attributs définis sur leurs domaines **D1,..., Dn** en **INTENSION**.

R est un sous-ensemble du produit cartésien **D1** × **D2** × **...** × **Dn**.

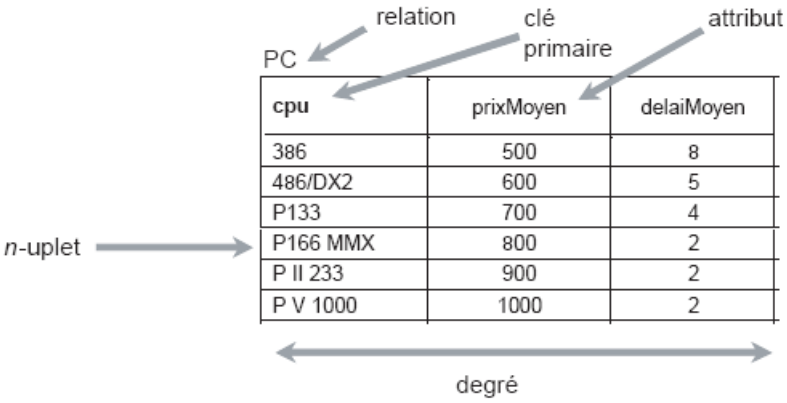
Pour travailler **formellement** avec le modèle relationnel, on considère que seuls **N** n-uplets de chaque relation sont présents dans la base (c'est ce qu'on appelle l'hypothèse du monde clos).

Définition en intention
Considérons l'exemple suivant : les micro-ordinateurs du marché sont caractérisés par le code du processeur, un prix moyen et un délai de livraison moyen. Les processeurs des plus anciens micro-ordinateurs sont des **386** et les plus récents sont des **Pentium V** cadencés à **1 GHz**. Les prix s'échelonnent de **500 F** à **1 000 F** et les délais de livraison varient de **0** à **8** jours.
La figure ci-dessous définit la relation en **intention** :

PC (cpu, prixmoyen, delaimoyen)

cpu prend sa valeur dans **D1** = {**'386'**, **'486/DX2'**, **'PIII 1000'**, …}
prixmoyen prend sa valeur dans **D2** = {**500**, **1000**}
delaimoyen prend sa valeur dans **D3** = {**0**, **8**}

Définition en extension
La définition en extension de la même relation est illustrée à la figure 2-2. Elle peut être considérée comme le contenu de cette relation. On voit bien qu'il s'agit d'un sous-ensemble du produit cartésien des trois domaines de valeurs. En effet, le tableau ne contient pas toutes les combinaisons de lignes qu'il est possible de composer avec les trois domaines de valeurs D1, D2 et D3.



Définition : Le **modèle relationnel** est un **modèle logique** associé aux **SGBD relationnels**.

Exemples : Oracle, **DB2**, **SQLServer**, **Access**, **Dbase**, etc.
Objectifs du modèle relationnel :

- *Indépendance physique* : indépendance entre **programmes** d'application et représentation interne de **données**.
- *Traitement des problèmes de **cohérence** et de **redondance** de données* : problème non traité au niveau des modèles **hiérarchiques** et **réseaux**.
- *Développement des LMD non procéduraux* : modélisation et manipulation simples de données, langages faciles à utiliser.
- *Devenir un standard*.

II. Concepts de base

II.1. Relation

Définition : Une relation **R** est un ensemble d'attributs **{**A1**,** A2**,**…**,** An**}**. Notation : **R (**A1, A2, ..., An**)**

Exemple : Soit la relation **PRODUIT (**numPd, designPd, puPd**)**
La relation **PRODUIT** est l'ensemble des attributs **{**numPd**,** designPd**,** puPd**}**

Remarque : Chaque attribut **Ai** prend ses valeurs dans un domaine **dom(Ai)**.
Exemple : Le prix unitaire **puPd** est compris entre **0** et **10000**. D'où **dom(puPd)** = **[0, 10000]**.

II.2. Tuple

Définition : Un **TUPLE** est un ensemble de valeurs t = <V1**,** V2**,** ... Vn> où **Vi** appartient à **dom(Ai)**. Il à noter que **Vi** peut aussi prendre la valeur nulle (NULL).

Exemple : **<2, 'voiture', 3000>** est un tuple

II.3. Contraintes d'intégrité

Définition : Une **CONTRAINTE D'INTEGRITE** est une **clause** permettant de **contraindre** la modification de tables, faite par l'intermédiaire de requêtes d'utilisateurs, afin que les **données saisies** dans la base soient conformes aux **données attendues**.

Types de contrainte d'intégrité :
- ***Clé primaire :*** Une CLE PRIMAIRE est un ensemble d'attributs dont les valeurs permettent de distinguer les tuples les uns des autres (identifiant).

Notation : la clé primaire doit être **soulignée**.

Une **contrainte** est de type CLE PRIMAIRE lorsqu'elle interdit l'existence de deux tuples ayant des valeur de clés identiques.

Exemple 1

```
CREATE TABLE R(
A1 INT(2),
A2 INT(2),
A3 INT(2),
PRIMARY KEY (A1)
)
INSERT INTO R VALUES (1, 2, 3);
INSERT INTO R VALUES (1, 4, 5);
```

R	A1	A2	A3
	1	2	3

Exemple 2:
Soit la relation **Produit (***numPd*, designPd, puPd**)**
L'attribut **numPd** présente la clé primaire de la relation Produit.

Une contrainte de type CLE PRIMAIRE sur la relation PRODUIT interdira l'existence de deux tuples ayant des valeurs de **numPD** identiques.

- ***Clé étrangère :*** attribut d'une relation **R1** qui est clé primaire d'une autre relation **R2**.
Notation : La clé étrangère doit être précédée par **#**.

Une **contrainte** est de type CLE ETRANGERE lorsque pour un tuple **T1** dans **R1**, elle vérifie l'existence d'un tuple **T2** dans **R2** ayant la même valeur de clés que **T1**.

Exemple 1:

```
CREATE TABLE R2(
        A21 INT(2),
        A22 INT(2),
        A23 INT(2),
        PRIMARY KEY (A21),
        FOREIGN KEY (A23) REFERENCES R(A1)
);

INSERT INTO R2 VALUES (1, 2, 3); // Erreur
```

INSERT INTO R2 VALUES (1, 2, **1**); // OK

R	A1	A2	A3
	1	2	3

R2	A21	A22	A23
	1	2	**1**

Exemple 2:
Considérons les relations :
FOURNISSEUR (*numFr*, rsFr, emailFr, adrFr**)** et
PRODUIT (*numPd*, designPd, puPd, **#numFr)**
Supposons qu'on a une **contrainte de type clé étrangère** sur la relation **PRODUIT.**
Alors l'existence du tuple T1 = **(12, 'Produit1', 100, 1)** dans la relation PRODUIT implique l'existence d'un tuple T2 dans la relation FOURNISSEUR dont l'attribut **numFr** a pour valeur **1**.

- ***Contraintes de domaine :*** les attributs doivent respecter une condition logique.

III. Traduction Modèle Entité/Association - Modèle relationnel

III.1. Règles

Pour traduire un modèle Entités / Associations en modèle relationnel, on applique les règles suivantes :

Règle N°1 - Chaque **entité** devient une **relation**. Les **attributs** de l'entité deviennent **attributs** de la relation. **L'identifiant** de l'entité devient **clé primaire** de la relation.

Règle N°2 - Chaque **association** de type **1-1** est prise en compte en incluant la **clé primaire** d'une des relations comme **clé étrangère** dans l'autre relation.

Règle N°3 - Chaque **association** de type **1-N** est prise en compte en incluant la clé primaire de la relation issue de **l'entité** dont la cardinalité maximale est N comme clé étrangère dans l'autre relation.

Règle N°4 Chaque association de type **M-N** est prise en compte en créant une nouvelle **relation** dont la clé primaire et la concaténation des clés primaires des relations issu des entités participantes. Les attributs de l'association sont insérés dans cette nouvelle relation.

III.2. Application

Exemple du modèle Entités/Associations élaboré dans le chapitre précédent (Chapitre 2, IV.2. Exemple illustratif) se traduit en modèle relationnel comme suit :

Règle N°1
CLIENT (numCl, nomCl, prenomCl, datenaisCl, adrCl**)**

Règle N°1 et Règle N°3
PRODUIT (numPd, designPd, puPd, **#numFr)**

Règle N°1
FOURNISSEUR (numFr, rsFr, emailFr, adrFr**)**

Règle N°4
COMMANDE (#numCl, #numPd, dateCd, qtePc)

IV. Les dépendances fonctionnelles (DF)

Le processus de **normalisation** permet de construire des bases de données relationnelles en évitant les **redondances** et en préservant **l'intégrité** des données. Il est préférable de normaliser les relations au moins jusqu'à la troisième forme normale (3FN).

La normalisation est basée sur les **dépendances fonctionnelles** (DF). **E.F. Codd** fut le premier à publier des écrits sur les DF [COD 72].

IV.1. Définitions

Définition :
Un attribut **B** dépend fonctionnellement d'un attribut **A** si à une valeur de **A** correspond au plus une valeur de **B**. La dépendance fonctionnelle est notée **A → B**.
Le membre droit de l'écriture s'appelle le **dépendant**, le membre gauche s'appelle le **déterminant**

Soit **R** un **schéma de relation** et **A** et **B** des attributs de **R**. On dit qu'une **instance r** de **R** satisfait la **FD X → Y** (on lit **A détermine fonctionnellement B**, ou simplement **A** détermine **B**.) si ce qui suit est vrai pour chaque paire de tuples **t1** et **t2** dans **r** :
Si **t1.X = t2.X**, alors **t1.Y = t2.Y**.

En d'autres termes :
Un attribut **B** dépend fonctionnellement d'un attribut **A** si à une valeur de **A** correspond au plus une valeur de **B**. Le membre droit de l'écriture s'appelle le **dépendant**, le membre gauche s'appelle le **déterminant**

Nous utilisons la notation **t1.A** pour faire référence à la projection du tuple **t1** sur l'attribut **A**.

Plusieurs attributs peuvent apparaître dans la **partie gauche** d'une DF. **B** est alors considéré comme un **ensemble** d'attributs. Dans ce cas, il convient de considérer le **couple** (si deux attributs figurent dans la partie gauche), le **triplet** (s'il y a trois attributs), etc.

Plusieurs attributs peuvent apparaître dans la **partie droite** d'une DF. **A** est alors considéré comme un **ensemble** d'attributs. Dans ce cas, il convient de considérer chaque DF en gardant la partie gauche et en faisant intervenir un seul attribut dans la partie droite.

Nous utilisons la notation **t1.A** pour faire référence à la projection du tuple **t1** sur les attributs de **A**

EXEMPLES

EXEMPLE 01
La figure ci-dessous illustre la signification de la FD **AB → C** en montrant une instance qui satisfait cette dépendance.

R	A	B	C	D
t1	**a1**	**b1**	**c1**	d1
t2	**a1**	**b1**	**c1**	d2
t3	a1	b2	c2	d1
t4	a2	b1	c3	d1

Les deux premiers tuples **t1** et **t2** montrent qu'un FD n'est pas la même chose qu'une **contrainte de clé** : bien que le FD ne soit pas violé, **AB** n'est clairement **pas une clé** pour la relation. Les troisième et quatrième tuples **t3** et **t4** illustrent que si deux tuples diffèrent soit dans le champ A soit dans le champ B, ils peuvent différer dans le champ C **sans violer** la FD.

D'un autre côté, si nous ajoutons un tuple **t5 = <a1, b1, c2, d1>** à l'instance montrée sur cette figure, l'instance résultante **violerait** la FD ; pour voir cette violation, comparez le premier tuple **t1** de la figure avec le nouveau tuple **t5**.

R	A	B	C	D
t1	**a1**	**b1**	**c1**	d1
t2	**a1**	**b1**	**c1**	d2
t3	a1	b2	c2	d1
t4	a2	b1	c3	d1
t5	**a1**	**b1**	**c1**	**d1**

Considérons les exemples suivants, qui concernent des pilotes ayant un **numéro**, un **nom**, une **fonction** (copilote , commandant, instructeur…) :

PILOTES (numPilote, nomPilote, fonction)
<**P0001**, **YAO**, **commandant**>
<**P0002**, **GNAGNE**, **copilote**>
<**P0003**, **OUATTARA**, **instructeur**>
<**P0004**, **SERY**, **instructeur**>
<**P0005**, **OUATTARA**, **copilote**>

- l'écriture **numPilote**, **jour** → **nbHeuresVol** est une DF, car à une occurrence du couple (**numPilote**, **jour**) correspond au plus une occurrence de **nbHeuresVol**;
- l'écriture **numPilote** → **nomPilote**, **fonction** est équivalente aux écritures **numPilote** → **nomPilote** et **numPilote** → **fonction** qui sont deux DF. En conséquence **numPilote** → **nomPilote**, **fonction** est une DF ;
- l'écriture **nomPilote** → **fonction** est une DF s'il n'y a pas **d'homonymes** dans la population des pilotes enregistrés dans la base de données. Dans le cas contraire, ce n'est pas une DF, car à un nom de pilote peuvent correspondre plusieurs fonctions ;
- l'écriture **fonction** → **nomPilote** n'est pas une DF, car à une fonction donnée correspondent éventuellement plusieurs pilotes.

Définition :
Une DF A, B → C est **ELEMENTAIRE** si ni A → C, ni B → C ne sont des DF.

EXEMPLES
Considérons les exemples suivants :

- la dépendance fonctionnelle **numPilote, jour** → **nbHeuresVol** est élémentaire, car **numPilote** → **nbHeuresVol** n'est pas une DF (un pilote vole différents jours, donc pour

un pilote donné, il existe plusieurs nombres d'heures de vol), pas plus que **jour → nbHeuresVol** (à un jour donné, plusieurs vols sont programmés) ;

- la dépendance fonctionnelle **numPilote, nomPilote → fonction** n'est pas élémentaire, car le numéro du pilote suffit pour retrouver sa fonction (**numPilote → fonction** est une DF).

Définition :
Une DF A → C est **DIRECTE** si elle n'est pas déduite par transitivité, c'est-à-dire s'il n'existe pas de DF A → B et B → C.

L'expérience montre qu'il est difficile de savoir si une DF est directe, mais il est aisé de voir si elle est indirecte. En conséquence si une DF n'est pas indirecte, elle est directe (ouf !).

EXEMPLE
Considérons l'exemple qui fait intervenir les attributs suivants :
immat : numéro d'immatriculation d'un avion ;
typeAvion : type de l'aéronef ;
nomConst : nom du constructeur de l'aéronef.

Avion2	immat	typeAvion	nomConst
	F-CLAR	CRJ	Canadian Regional Jet
	F-ROMA	A320	Airbus
	F-GLDX	A320	Airbus
	F-CSTU	B727	Boeing
	F-STEF	A330	Airbus
	F-PAUL	B747	Boeing
	F-ABDL	A340	Airbus

- La dépendance immat → nomConst n'est pas une DF directe.
- La dépendance immat → typeAvion est une DF directe.
- La dépendance typeAvion → nomConst est une DF directe.

Remarque : Il est essentiel de bien remarquer qu'une dépendance fonctionnelle (en abrégé, DF) est une assertion sur **toutes les valeurs possibles** et non pas sur les valeurs actuelles : elle caractérise une **intention** et non pas une **extension** d'une relation.

IV.2. Propriétés des dépendances fonctionnelles

Soient X, Y et Z des ensembles d'attributs. On notera XY l'ensemble X ∪ Y. Les dépendances fonctionnelles obéissent à certaines propriétés connues sous le nom d'axiomes d'Armstrong.
- **Réflexivité** : X →X
- **Augmentation** : X → Y ⇒ XZ → YZ
- **Transitivité** : X → Y et Y → Z ⇒ X → Z

D'autres propriétés se déduisent de ces axiomes :
- **Union** : X → Y et X → Z ⇒ X → YZ
- **Pseudo-transitivité** : X → Y et YW → Z ⇒ XW → Z
- **Décomposition** : X → Y et Z ⊆ Y ⇒ X → Z

L'intérêt de ces axiomes et des propriétés déduites est de pouvoir construire, à partir d'un premier ensemble de dépendances fonctionnelles, l'ensemble de **toutes les dépendances fonctionnelles** qu'elles génèrent.

IV.3. Fermeture d'un ensemble de DF

Définition :
La **FERMETURE F$^+$** d'un ensemble **F** de DF est l'ensemble des DF qu'on peut obtenir par applications successives des axiomes d'Armstrong.

Définition :
La **COUVERTURE MINIMALE** est l'ensemble C des DF **élémentaires** issues de F+ tel que :
- le membre droit (dépendant) de chaque DF ne contient qu'un seul attribut ;
- le membre gauche (déterminant) de chaque DF est irréductible ;
- aucune DF ne peut être supprimée.

En considérant seulement
F = {A → B**,** B → C**}**

La **FERMETURE** obtenue contient **21** DF

F$^+$ = {
A → A, reflexivité
B → B, reflexivité
A → C,
A, C → B,
…
etc.
}.

L'exemple suivant décrit la résolution de la **FERMETURE** et de la **COUVERTURE MINIMALE** de l'ensemble de DF.
F = {
immat → compagnie,
immat → typeAvion,
typeAvion → capacite,

typeAvion → nomConst
}.

On applique la **transitivité** entre :

- la DF **immat → typeAvion** et **typeAvion → nomConst** et on obtient la DF **immat → nomConst** ;
- la DF **immat → typeAvion** et **typeAvion → capacite** et on obtient la DF **immat → capacite**.

On pourrait appliquer **l'union** entre **immat → compagnie** et **immat → typeAvion** de manière à obtenir la DF **immat → typeAvion, compagnie**.
De même, **l'union** entre **typeAvion → capacite** et **typeAvion → nomConst** donne **typeAvion → capacite, nomConst**.

On pourrait appliquer **l'augmentation** entre chaque DF. Ainsi la DF immat → compagnie donnerait lieu à l'écriture des DF suivantes :

- immat, typeAvion → compagnie, typeAvion
- immat, capacite → compagnie, capacite
- immat, nomConst → compagnie, nomConst

On pourrait également appliquer les autres propriétés des DF aux DF initiales de manière à obtenir un **nombre important de DF pas nécessairement intéressantes**. La fermeture de cet ensemble s'écrirait de la sorte :

$\mathbf{F^+}$ = {immat → compagnie ; immat → typeAvion ; typeAvion → capacite ; typeAvion → nomConst ; immat → capacite ; immat → nomConst ; immat → typeAvion, compagnie ; immat, typeAvion → compagnie, typeAvion ; immat,capacite → compagnie, capacite ; immat, nomConst → compagnie, nomConst ; … }

Ce qui intéresse le concepteur est l'ensemble des DF composant la **COUVERTURE MINIMALE** de $\mathbf{F^+}$ noté

C = {
immat → compagnie ;
immat → typeAvion ;
typeAvion → capacite ;
typeAvion → nomConst
}.

Il existe des méthodes de conception de modèle E/A basées sur les DF.
Alors que les DF vont servir à **classifier** un schéma relationnel en **première**, **deuxième**, **troisième** ou **BCFN** (Boyce-Codd forme normale), d'autres formes de dépendances vont permettre de définir les **quatrième** et **cinquième** formes normales. Ces familles de dépendances sont les **dépendances multivaluées** et les **dépendances de jointure**.

Néanmoins, la **majorité** des schémas relationnels en **entreprise** sont en **deuxième** (on dénormalise des relations volontairement pour des contraintes d'optimisation) ou en **troisième** forme normale.

V. Normalisation

Étant donné un schéma de relations, nous devons décider s'il s'agit d'une **bonne conception** ou si nous devons le **décomposer en relations plus petites**. Une telle décision doit être guidée par une compréhension des problèmes, le cas échéant, qui découlent du schéma actuel. Pour fournir une telle orientation, plusieurs formes normales ont été proposées. Si un schéma relationnel se présente sous l'une de ces formes normales, nous savons que certains types de problèmes ne peuvent pas survenir.

Les formes normales basées sur les FD sont la première forme normale (1NF), la deuxième forme normale (2NF), la troisième forme normale (3NF) et la forme normale Boyce-Codd (BCNF). Ces formes ont des exigences de plus en plus restrictives : toute relation en BCNF est aussi en 3NF, toute relation en 3NF est aussi en 2NF, et toute relation en 2NF est en 1NF.

V.1. Objectifs de la normalisation

Exemple : Soit la relation **COMMANDE_PRODUIT (**NumProd, Quantite, NumFour, Adresse**)**.

COMMANDE_PRODUIT	NumProd	Quantite	NumFour	Adresse
	101	300	**901**	**Rue 12 treichville**
	104	1000	902	Av 7 Décembre
	112	78	904	Rue 20 Mars
	103	250	**901**	**Rue 12 treichville**
			1000	**Rue 11 koumassi**

Cette relation présente différentes anomalies.

- *Anomalies de **modification*** : Si l'on souhaite mettre à jour l'adresse d'un fournisseur, il faut le faire pour tous les tuples concernés.
- *Anomalies **d'insertion*** : Pour ajouter un nouveau fournisseur, il faut obligatoirement fournir des valeurs pour **NumProd** et **Quantité**.
- *Anomalies de **suppression*** : La suppression du produit **104** fait perdre toutes les informations concernant le fournisseur **902**.

Pour faire face à ce genre de problèmes, on a recours à la **normalisation**.
Objectifs de la normalisation :

- *Suppression des problèmes de mise à jour*
- *Minimisation de l'espace de stockage (élimination des redondances)*

V.2. Première forme normale (1FN)

Définition :
Une relation est en **1FN** si **tout attribut est atomique** (n'est pas décomposable en d'autres attributs).
En d'autres termes, au niveau de l'extension d'une relation, à l'intersection d'une ligne et d'une colonne, on ne doit trouver qu'**une et une seule valeur** (qui peut être diverse : nombre, chaîne de caractères, date, image, etc.). c'est-à-dire pas des **listes** ou des **ensembles**

Bien que certains des systèmes de bases de données les plus récents assouplissent cette exigence, dans ce chapitre nous supposerons qu'elle est toujours valable.

EXEMPLE
La relation **Vols1** ci-dessous respecte la règle de la première forme normale. À l'intersection de chaque ligne et pour chaque colonne il n'existe **qu'une seule valeur**. Il y a cependant des **redondances** (nom de la compagnie, sexe du pilote et type de l'aéronef).

Dans cet exemple, des pilotes peuvent voler pour le compte de différentes compagnies.
Si on devait définir une **clé** pour la relation **Vols1**, ce serait la concaténation de **ncomp**, **nomPilote**, **immat**. Cette combinaison permet en effet d'identifier chaque ligne.
Le nombre d'heures de vol dépend au minimum des valeurs de ces trois attributs.

Vols1	ncomp	compagnie	nomPilote	sexe	typeAvion	immat	nbHeuresVol
	1	Air-France	Bidal	F	CRJ	F-CLAR	600
	1	Air-France	Bidal	F	A320	F-ROMA	345
	1	Air-France	Bidal	F	A320	F-GLDX	120
	1	Air-France	Bidal	F	B727	F-CSTU	150
	1	Air-France	Labat	F	A320	F-GLDX	70
	1	Air-France	Labat	F	A320	F-ROMA	340
	1	Air-France	Labat	F	B727	F-CSTU	9000
	2	Quantas	Sanfilippo	G	A320	F-STEF	7500
	2	Quantas	Sanfilippo	G	A320	F-GLDX	250
	2	Quantas	Bidal	F	B727	F-CSTU	2500
	2	Quantas	Soutou	G	B747	F-PAUL	750
	2	Quantas	Soutou	G	A320	F-ABDL	2500

Les tables relationnelles sont nativement toutes en **première forme normale**, car les attributs de type tableau ne sont pas autorisés au niveau de la base de données.

V.3. Deuxième forme normale (2FN)

Définition :
Une relation est en (2FN) si elle est en première forme normale, d'une part, et si tout attribut n'appartenant pas à la **clé** primaire est en **dépendance fonctionnelle ELEMENTAIRE** avec la clé, d'autre part.

En d'autres termes :
Une relation est en 2FN si :
1) elle est en 1FN ;
2) tout attribut **non clé primaire** dépend **ENTIEREMENT** de la clé primaire.

EXEMPLE1

Avion2	immat	typeAvion	nomConst
	F-CLAR	CRJ	Canadian Regional Jet
	F-ROMA	A320	Airbus
	F-GLDX	A320	Airbus
	F-CSTU	B727	Boeing
	F-STEF	A330	Airbus
	F-PAUL	B747	Boeing

F-ABDL	A340	Airbus

La relation **Avions2** n'est pas en **deuxième forme normale**, car la dépendance **immat → nomConst n'est pas une DF directe**. Car on a les Df suivantes :

immat → typeAvion

et

typeAvion → nomConst (car Il suffit en effet de connaître le type de l'aéronef pour en déduire le constructeur)

Les DF **immat → typeAvion** et **typeAvion → nomConst** sont directes.

EXEMPLE2
Soient les relations suivantes :

CLIENT (NumCli, Nom, Prénom, DateNaiss, Rue, CP, Ville**)**

Et

COMMANDE_PRODUIT(NumProd, Quantite, **NumFour**, Ville)

La relation CLIENT **est en 1FN**. Tout attribut non clé primaire depend entièrement de la clé primaire **NumCli. On conclut que la relation CLIENT est en 2FN**

La relation **COMMANDE_PRODUIT est en 1FN**.
On a la DF : **NumFour → Ville**.
Or **NumFour** est une partie de {**NumProd, NumFour**} qui est la clé. On conclut que la relation **COMMANDE_PRODUIT n'est pas en 2FN**.

La décomposition suivante de la relation **COMMANDE_PRODUIT** donne deux relations en **2FN** :

COMMANDE_PRODUIT(NumProd, Quantite, **NumFour**, Ville)

COMMANDE (NumProd, **#NumFour**, Quantité**)** ;

FOURNISSEUR (NumFour, Ville**)**.

V.4. Troisième forme normale (3FN)

Définition :
Une relation est en (3FN) si elle est en (2FN) et si les dépendances fonctionnelles entre la **clé primaire** et les attributs ne faisant pas parti de la clé sont directes.
En d'autres termes :
Une relation est en 3FN si :
1) elle est en 2FN ;
2) **il n'existe aucune DF** entre deux attributs **non clé** primaire de cette relation.

EXEMPLE

La relation **COMPAGNIE (Vol**, Avion, Pilote**)**
avec les **DF** :

- Vol → Avion,
- **Avion → Pilote** (**DF** entre deux attributs non clé primaire) et
- Vol → Pilote

est en **2FN**, mais pas en **3FN**.

COMPAGNIE	**Vol**	**Avion**	**Pilote**
	V0001	F-CLAR	Gnagne
	V0002	F-CSTU	Sery
	V0003	F-GLDX	Mamadou
	V0004	F-CSTU	Konan
	V0005	F-GLDX	Mamadou
	V0006	F-ROMA	Sery
	V0007	F-CSTU	Konan

Anomalies de mise à jour sur la relation **COMPAGNIE** : Il n'est pas possible de saisir un **nouvel** avion sur un **nouveau** vol sans préciser le pilote correspondant.

La **décomposition** suivante donne deux relations en **3FN** qui permettent de retrouver (par transitivité) toutes les DF :

COMPAGNIE (Vol, Avion, Pilote**)**

R1 (Vol, #**Avion)** ;

R2 (Avion, Pilote**)**.

Chapitre 4 : L'algèbre relationnelle

Objectifs spécifiques

A la fin de ce chapitre, l'étudiant doit être capable de :
- Reconnaître l'utilité des **opérateurs** ensemblistes et spécifiques
- Analyser des **requêtes** plus ou moins complexes
- Appliquer les **opérateurs** appropriés dans l'expression des **requêtes**

Plan du chapitre

I. Définition
II. Opérateurs ensemblistes
III. Opérateurs spécifiques
IV. Exercice d'application

I. Définition

L'algèbre relationnelle est définie comme étant l'ensemble **d'opérateurs** qui s'appliquent aux **relations**.
Résultat : **nouvelle relation** qui peut à son tour être manipulée.
L'algèbre relationnelle permet **d'effectuer des recherches** dans les **relations**.

II. Opérateurs ensemblistes

II.1. Union

Définition :
L'union des relations R1(a1,..., an) et R2(b1,..., bn) (ayant même structure) donne la relation R3 qui contient **les tuples de R1** et **les tuples de R2 qui n'appartiennent pas à R1** ;
elle est notée R3= **U(**R1**,** R2**)** OU R3 = R1 **U** R2 ou R3 = **UNION(**R1**,** R2**)**.

EXEMPLE : Pilote = PiloteAF U PiloteSING

PiloteAF	brevet	nom	nbHVol
	PL-1	Christian Soutou	450
	PL-2	Fréderic Brouard	900
	PL-5	Pierre Sery	670
	PL-4	Gille Laborde	1500

PiloteSING	brevet	nom	nbHVol
	PL-3	Pierre Sery	1000
	PL-4	Gille Laborde	1500

Pilote	brevet	nom	nbHVol
	PL-1	Christian Soutou	450
	PL-2	Fréderic Brouard	900
	PL-5	Pierre Sery	670
	PL-4	Gille Laborde	1500
	PL-3	Pierre Sery	1000

II.2. Intersection

Définition :
L'intersection des relations R1(a1,..., an) et R2(b1,..., bn) (ayant même structure) donne la relation R3 contenant les tuples qui appartiennent à la fois à R1 et à R2.
elle est notée R3= ∩ **(**R1**,** R2 **)** ou R3=R1 ∩ R2 ou R3=**INTERSECT(**R1**,** R2**)**.

EXEMPLE : Pilote = PiloteAF ∩ PiloteSING

PiloteAF	brevet	nom	nbHVol
	PL-1	Christian Soutou	450
	PL-2	Fréderic Brouard	900
	PL-5	Pierre Sery	670

PiloteSING	brevet	nom	nbHVol
	PL-5	Pierre Sery	670
	PL-4	Gilles Laborde	1500

Pilote	brevet	nom	nbHVol
	PL-5	Pierre Sery	670

II.3. Différence

Définition :
La différence des relations R1(a1,..., an) et R2(b1,..., bn) (ayant même structure) donne la relation R3 qui contient les tuples de R1 qui n'appartiennent pas à R2 ;
elle est notée R3= **–(**R1**,** R2**)** ou R3=R1 **–** R2 ou R3=**MINUS(**R1**,** R2**)**.

EXEMPLE : Pilote = PiloteAF – PiloteSING

PiloteAF	brevet	nom	nbHVol
	PL-1	Christian Soutou	450
	PL-2	Fréderic Brouard	900
	PL-3	Pierre Sery	670

PiloteSING	brevet	nom	nbHVol
	PL-3	Pierre Sery	670
	PL-4	Gille Laborde	1500

Pilote	brevet	nom	nbHVol
	PL-1	Christian Soutou	450
	PL-2	Fréderic Brouard	900

II.4. Division

Définition :
La division de R1(a1,...,an,b1,...,bn) par R2[b1,...,bn] donne la relation R3[a1,...,an] qui contient tous les tuples tels que la concaténation à chacun des tuples de R2 donne toujours un tuple de R1 ;
elle est notée R3 = ÷ [R1,R2] ou R3 = R1 ÷ R2 ou R3 = DIVISION(R1,R2)

EXEMPLE : TouteComp = Affreter ÷ Compagnie

Affreter	immat	typeAv	comp
	A1	A320	SING
	A2	A340	AF
	A3	**Mercure**	**AF**
	A4	A330	ALIB
	A3	**Mercure**	**ALIB**
	A3	**Mercure**	**SING**

Compagnie	comp
	AF
	ALIB
	SING

TouteComp	immat	typeAv

Visualisation de la division

A		B		A ÷ B

II.5. Produit cartésien

Définition :
Le produit cartésien de R1(a1,…,an) par R2(b1,…,bn) donne la relation R3(a1,…,an,b1,…,bn) qui contient les combinaisons des tuples de R1 et de R2 ;
il est notée R3 = × (R1, R2) ou R3 = R1 × R2 ou R3 = **PRODUCT**(R1, R2).

EXEMPLE : PiloteEtAvion = Pilote × Avion

Pilote	brevet	nom	compa
	PL-1	Gratien Viel	AF
	PL-2	Richard Grin	SING
	PL-3	Placide Fresnais	AF

Avion	immat	typeAvion	nbHVol
	F-WTSS	Concorde	6570
	F-GLFS	A320	3500

PiloteEtAvion	brevet	nom	compa	immat	typeAvion	nbHVol
	PL-1	Gratien Viel	AF	F-WTSS	Concorde	6570
	PL-2	Richard Grin	SING	F-WTSS	Concorde	6570
	PL-3	Placide Fresnais	AF	F-WTSS	Concorde	6570
	PL-1	Gratien Viel	AF	F-GLFS	A320	3500
	PL-2	Richard Grin	SING	F-GLFS	A320	3500
	PL-3	Placide Fresnais	AF	F-GLFS	A320	3500

II.6. Renommage

Définition :
R3 = α(R1, R2) ou R3 = **RENAME**(R1, R2). R3 est la relation obtenue en renommant R1 en R2.

III. Opérateurs spécifiques

III.1. Projection

Définition :
La projection de la relation R1[A1,...,An] sur les attributs Ai,...,Am est une relation R2[Ai,...,Am], qui est notée R2=$\Pi_{<Ai, ..., Am>}$(R1) ou R2 = **PROJECT**(R / Ai, …, Am).
R2 contient seulement les **données** présentes dans les colonnes **Ai**,...,**Am** issues de **R1**.

EXEMPLE :

$$PiloteExp = \prod_{<nom,\ nbHVol>} (Pilote)$$

PiloteAF	brevet	nom	nbHVol	comp
	PL-1	Christian Soutou	450	AF
	PL-2	Fréderic Brouard	900	AF
	PL-3	Paul Soutou	1000	SING
	PL-4	Gile Laborde	1500	SING
	PL-5	Pierre Sery	670	AF

PiloteExp	nom	nbHVol
	Christian Soutou	450
	Fréderic Brouard	900
	Paul Soutou	1000
	Gile Laborde	1500
	Pierre Sery	670

Visualisation de la projection

$$R2 = \prod_{<A2,\ A4,\ A6>} (R1)$$

R1	A1	A2	A3	A4	A5	A6	A7

R2	A2	A4	A6

III.2. Restriction

Définition :
La restriction de la relation R1(A1,…,An), selon la condition *C* est une relation R2(A1,…,An)
qui est notée $R2 = \sigma_{<C>}(R1)$ ou R2=**RESTRICT(**R1/C**)**. R2 contient les tuples de R1 qui satisfont la condition C.

EXEMPLE :

PiloteAF $= \sigma_{<\text{Compta='AF'}>}(Pilote)$

Pilote	brevet	nom	nbHVol	comp
	PL-1	Christian Soutou	450	AF
	PL-2	Fréderic Brouard	900	AF
	PL-3	Paul Soutou	1000	SING
	PL-4	Gile Laborde	1500	SING
	PL-5	Pierre Sery	670	AF

PiloteAF	brevet	nom	nbHVol	comp
	PL-1	Christian Soutou	450	AF
	PL-2	Fréderic Brouard	900	AF
	PL-5	Pierre Sery	670	AF

III.3. Jointure

Définition :
La jointure de R1(**A1**,…,**An**) avec R2[**B1**,…,**Bn**] suivant la condition *C* donne la relation R3[**A1**,…,**An**, **B1**,…,**Bn**], **produit cartesien** de R1 et R2 qui contient les **tuples vérifiant la condition C** ;
elle est notée : R3 = R1 $\triangleright\triangleleft_{(C)}$ R2 ou R3 = $\triangleright\triangleleft_{(C)}$ [R1, R2]

EXEMPLE : PiloteAvion = Pilote ▷◁(avi=immat) Avion

Pilote	brevet	nom	avi
	PL-1	Gratien Viel	F-WTSS
	PL-2	Richard Grin	
	PL-3	Placide Fresnais	F-WTSS

Avion	immat	typeAvion	nbHVol
	F-WTSS	Concord	6570
	F-GLFS	A320	3500

Pilote × Avion	brevet	nom	avi	immat	typeAvion	nbHVol
	PL-1	Gratien Viel	F-WTSS	F-WTSS	Concord	6570
	PL-2	Richard Grin		F-WTSS	Concord	6570
	PL-3	Placide Fresnais	F-WTSS	F-WTSS	Concord	6570
	PL-1	Gratien Viel	F-WTSS	F-GLFS	A320	3500
	PL-2	Richard Grin		F-GLFS	A320	3500
	PL-3	Placide Fresnais	F-WTSS	F-GLFS	A320	3500

PiloteAvion	brevet	nom	avi	immat	typeAvion	nbHVol
	PL-1	Gratien Viel	F-WTSS	F-WTSS	Concorde	6570
	PL-3	Placide Fresnais	F-WTSS	F-WTSS	Concorde	6570

IV. Exercice d'application

Soit le modèle relationnel suivant :

CLIENT (**NOC**, **NOM**, **ADRESSE**)

CLIENT	NOC	NOM	ADRESSE
	C001	KONE	Abidjan
	C002	SERY	Bouaké
	…		

SERVICE (NOS, INTITULE, LOCALISATION)

SERVICE	NOS	INTITULE	LOCALISATION
	S001	Service Informatique	Daloa
	S002	Service Comptabilité	Man
	…		

PIECE (NOP, DESIGNATION, COULEUR, POIDS)

PIECE	NOP	DESIGNATION	COULEUR	POIDS
	P001	Carte Mère	verte	50g
	P002	Micro-processeur	gris	3g
	…		rouge	

ORDRE (#NOP, #NOS, #NOC, QUANTITE)

ORDRE	#NOP	#NOS	#NOC	QUANTITE
	P001	S001	C001	500
	P002	S001	C002	100
	…			

Proposer, en **algèbre relationnelle**, une formulation des **requêtes** suivantes.

1. Déterminer les **NOS** des services qui ont commandé pour le client **C001**.
2. Déterminer les **NOS** des services qui ont commandé une pièce **P001** pour le client **C001**.
3. Donner les **NOS** des services qui ont commandé une pièce de couleur "**rouge**" pour le client **C001**.
4. Donner les **NOC** des clients qui ont en commande au moins toutes les pièces commandées par le service **S001**.
5. Donner les **NOC** des clients qui ont en commande des pièces figurant uniquement dans les commandes du service **S001**.
6. Donner les **NOP** des pièces qui sont commandées au **niveau local** (pour ces pièces, **l'adresse** du client et la **localisation** du service sont identiques).

Chapitre 5 : Le langage SQL

Objectifs spécifiques

A la fin de ce chapitre, l'étudiant doit être capable de :

- Apprendre à **créer** une **base de données** en tenant compte des **contraintes d'intégrité**
- Savoir **ajouter**, **modifier**, **supprimer** des enregistrements d'une table
- Construire des **requêtes d'interrogations** correspondant à des critères plus ou moins complexes
- Appliquer des **droits d'accès** à une base de données

Plan du chapitre

I. Présentation de SQL

SQL signifie **Structured Query Language.**Ilest le langage des bases de données relationnelles répandant à la fois aux problématiques de création des objets de base de données (modèle), de manipulation des données (algèbre relationnelle), de gestion de la sécurité (droits d'accès), de traitements locaux de données (procédures). Il s'agit d'un langage non procédural qui a été conçu par IBM dans les années 70. Il est devenu le langage standard des systèmes de gestion de bases de données relationnelles (SGBDR) depuis 1986. Il est utilisé par les principaux SGBDR du marché : Oracle, SQL Server, MySQL, Access, DB2, etc.

Remarque : Il existe plusieurs implémentations de SQL chez les principaux éditeurs.
Dans le reste de ce chapitre, l'implémentation utilisée est celle du SGBD Oracle.

II. Définition de données

II.1. Création des tables

Syntaxe : Pour créer une table, on fait recours à l'instruction suivante :

```
CREATE TABLE nom_table
(
        Attribut1 type1,
        Attribut2 type2,
        …,
        Contrainte1,
        Contrainte2,
        …
) ;
```

Exemple
Dans WampSercer, créer la base de données M1IABD

```
CREATE TABLE Employee (
        Employee_id int AUTO_INCREMENT PRIMARY KEY,
        First_name VARCHAR(50),
        Last_name VARCHAR(50),
        Salary int,
        Joining_date Date,
        Departement VARCHAR(50)
);
```

```
INSERT INTO Employee (Employee_id, First_name, Last_name, Salary, Joining_date, Departement) VALUES (1, 'Bob', 'Kinto', 1000000, "2019-01-20", "Finance");
INSERT INTO Employee (Employee_id, First_name, Last_name, Salary, Joining_date, Departement) VALUES (2, 'Jerry', 'Kansxo', 6000000, "2019-01-15", "IT");
INSERT INTO Employee (Employee_id, First_name, Last_name, Salary, Joining_date, Departement) VALUES (3, 'Philip', 'Jose', 8900000, "2019-02-05", "Banking");
INSERT INTO Employee (Employee_id, First_name, Last_name, Salary, Joining_date, Departement) VALUES (4, 'John', 'Abraham', 2000000, "2019-02-25", "Insurance");
```

```
INSERT INTO Employee (Employee_id, First_name, Last_name, Salary, Joining_date, Departement) VALUES (5, 'Michael', 'Mathew', 2200000, "2019-02-28", "Finance");
INSERT INTO Employee (Employee_id, First_name, Last_name, Salary, Joining_date, Departement) VALUES (6, 'Alex', 'chreketo', 4000000, "2019-05-10", "IT");
INSERT INTO Employee (Employee_id, First_name, Last_name, Salary, Joining_date, Departement) VALUES (7, 'Yohan', 'Soso', 1230000, "2019-06-20", "Banking");
```

Employee	Employee_id	First_name	Last_name	Salary	Joining_date	Departement
	1	Bob	Kinto	1000000	2019-01-20	Finance
	2	Jerry	Kansxo	6000000	2019-01-15	IT
	3	Philip	Jose	8900000	2019-02-05	Banking
	4	John	Abraham	2000000	2019-02-25	Insurance
	5	Michael	Mathew	2200000	2019-02-28	Finance
	6	Alex	chreketo	4000000	2019-05-10	IT
	7	Yohan	Soso	1230000	2019-06-20	Banking

Type des données :

- NUMBER(N) : Entier à N chiffres
- NUMBER(N , M) : Réel à N chiffres au total, M après la virgule.
- DATE : Date complète (date et/ou heure)
- VARCHAR(N), VARCHAR2(N) : chaîne de N caractères (entre ' ') dont les espaces enfin de la chaîne seront éliminés (longueur variable).
- CHAR(N) : Chaîne de N caractères (longueur fixe).

Contraintes d'intégrité

Définition : Dans la définition d'une table, on peut indiquer des contraintes d'intégrité portant sur une ou plusieurs colonnes.

Les contraintes possibles sont :

- UNIQUE,
- PRIMARY KEY,
- FOREIGN KEY REFERENCES et
- CHECK.

Chaque contrainte doit être **nommée** pour pouvoir la mettre à jour ultérieurement.

Exemple de clé étrangère

Supposons que chaque personne a effectué des commandes. Pour stocker les commandes, vous pouvez créer une nouvelle table nommée « Commandes »:

```
CREATE TABLE Commandes (
    CommandeID int AUTO_INCREMENT PRIMARY KEY,
    NumCommande int NOT NULL,
    PersonneID int,
    FOREIGN KEY (PersonneID) REFERENCES Personnes(PersonneID)
);
```

La colonne « PersonneID » est une clé étrangère qui fait référence à la colonne « PersonneID » de la table « Personnes ». Nous avons utilisé la contrainte « Foreign Key » pour établir cette relation:

```
FOREIGN KEY (PersonneID) REFERENCES Personnes(PersonneID)
```

Création :

- **CONSTRAINT** nom_contrainte **UNIQUE** (colonne1, colonne2, …) : interdit qu'une colonne, ou la concaténation de plusieurs colonnes, contiennent deux valeurs identiques.
- **CONSTRAINT** nom_contrainte **PRIMARY KEY** (attribut1, attribut2, …) : l'ensemble des attributs attribut1, attribut2, … forment la clé primaire de la relation.
- **CONSTRAINT** nom_contrainte **FOREIGN KEY** (attribut_clé_étrangère) **REFERENCES** nom_table (attribut_référence) : l'attribut de la relation en cours représente la clé étrangère qui fait référence à la clé primaire de la table indiquée.
- **CONSTRAINT** nom_contrainte **CHECK** (condition) : contrainte là où on doit obligatoirement satisfaire la condition telle qu'elle est énoncée.

***- Exemple* :**

```
CREATE TABLE reward (
    Employee_ref_id int,
    date_reward Date,
    amount int,
    FOREIGN KEY (Employee_ref_id) REFERENCES Employee(Employee_id)
);
```

```
INSERT INTO reward (Employee_ref_id, date_reward, amount)
VALUES (1, '2019-05-11', '1000');
INSERT INTO reward (Employee_ref_id, date_reward, amount)
```

```
VALUES (2, '2019-02-15', '5000');
INSERT INTO reward (Employee_ref_id, date_reward, amount)
VALUES (3, '2019-04-22', '2000');
INSERT INTO reward (Employee_ref_id, date_reward, amount)
VALUES (1, '2019-06-20', '8000');
```

reward	Employee_ref_id	date_reward	amount
	1	2019-05-11	1000
	2	2019-02-15	5000
	3	2019-04-22	2000
	1	2019-06-20	8000

L'insertion de l'enregistrement suivant sera refusée.

```
INSERT INTO reward (Employee_ref_id, date_reward, amount)
VALUES (8, '2019-06-20', '8000');
```

II.2. Renommage des tables

Syntaxe : Pour changer le nom d'une table, on fait recours à l'instruction suivante :

```
RENAME Ancien_Nom
TO Nouveau_Nom ;
```

Exemple : Etant donné l'entité ETUDIANT, si on souhaite la renommer par STUDENT, on écrit :

```
RENAME ETUDIANT
TO STUDENT ;
```

II.3. Destruction des tables

Syntaxe : Pour supprimer le contenu d'une table ainsi que son schéma, on utilise l'instruction qui suit :

```
DROP TABLE nom_table ;
```

Remarque : Attention, la suppression d'une table engendre la perte des données qu'elle contient.

Exemple : Pour supprimer la table ETUDIANT ainsi que son contenu, on fait recours à l'instruction :

```
DROP TABLE ETUDIANT ;
```

II.4. Modification des tables

Il existe plusieurs modifications que l'on peut effectuer sur une table donnée.
Ajout d'attributs : Après avoir créé la base de données, des tâches de maintenance semblent être parfois nécessaires. D'où l'ajout d'un nouvel attribut :

```
ALTER TABLE nom_table
ADD (attribut type, …) ;
```

Exemple :
Etant donné la table Commande, l'ajout du champ Montant à cette table revient à écrire :

```
ALTER TABLE COMMANDE
ADD (MONTANT NUMBER(10,3)) ;
```

Modification des attributs : Après avoir créé la base de données, on peut modifier le type d'un attribut en utilisant l'instruction suivante :

```
ALTER TABLE nom_table
MODIFY (attribut type, …) ;
```

Exemple :
Modifier le nombre de chiffres du champ Montant de la table Commande nécessite le recours à l'instruction :

```
ALTER TABLE COMMANDE
MODIFY (MONTANT NUMBER(12,3)) ;
```

Ajout de contraintes : Après avoir créé la base de données, on peut ajouter une nouvelle contrainte d'intégrité grâce à l'instruction suivante :

```
ALTER TABLE nom_table
ADD CONSTRAINT nom_contraintedefinition_contrainte ;
```

Exemple : Ajouter une contrainte à la table Commande qui permet d'obliger des insertions de montants positifs

ALTER TABLE COMMANDE
ADD CONSTRAINT CK_MONTANT **CHECK(**MONTANT >= 0**)** ;

Suppression de contraintes : Pour supprimer une contrainte, on procède comme indique la syntaxe de cette instruction :

```
ALTER TABLE nom_table
DROP CONSTRAINT nom_contrainte ;
```

Exemple : Supprimer la contrainte ck_montant de la table Commande

ALTER TABLE COMMANDE
DROP CONSTRAINT CK_MONTANT ;

III. Manipulation de données

III.1. Ajout de données

Syntaxe : Pour ajouter un tuple dans une table, on procède comme suit :

```
INSERT INTO nom_table
VALUES (valeur_attribut1, valeur_attribut2, .. ) ;
```

Exemple : Etant donné la table Etudiant(NCE, nom, prenom, ville). Si on souhaite insérer les informations d'un nouvel étudiant disposant des informations suivantes
(1234, sery , yves, Bouaké), on écrit :

INSERT INTO ETUDIANT
VALUES (1234, "SERY" ,"YVES", "BOUAKE")

III.2. Modification de données

Syntaxe : Pour modifier la valeur d'un attribut relatif à un ou plusieurs tuples d'une table, on procède comme suit :

```
UPDATE nom_table
SET attribut1 = valeur1, attribut2 = valeur2,
[WHERE condition] ;
```

Exemple : Etant donné la table Etudiant (NCNI, nom, prenom, ville). Si jamais l'étudiant sery de numéro CNI 123 habite maintenant à Abidjan, on écrit dans ce cas :

UPDATE ETUDIANT
SET VILLE= "Abidjan"
WHERE NCE=1234 ;

Exemple:
Considérons la table **Employee** suivante

Employee	Employee_id	First_name	Last_name	Salary	Joining_date	Departement
	1	Bob	Kinto	1000000	2019-01-20	Finance
	2	Jerry	Kansxo	6000000	2019-01-15	IT
	3	Philip	Jose	8900000	2019-02-05	Banking
	4	John	Abraham	2000000	2019-02-25	Insurance
	5	Michael	Mathew	2200000	2019-02-28	Finance
	6	Alex	chreketo	4000000	2019-05-10	IT
	7	Yohan	Soso	1230000	2019-06-20	Banking

```
UPDATE Employee
SET First_name ='ADAMA'
WHERE Employee_id = 1;
```

Employee	Employee_id	First_name	Last_name	Salary	Joining_date	Departement
	1	KONAN	Kinto	1000000	2019-01-20	Finance
	2	Jerry	Kansxo	6000000	2019-01-15	IT
	3	Philip	Jose	8900000	2019-02-05	Banking
	4	John	Abraham	2000000	2019-02-25	Insurance
	5	Michael	Mathew	2200000	2019-02-28	Finance
	6	Alex	chreketo	4000000	2019-05-10	IT
	7	Yohan	Soso	1230000	2019-06-20	Banking

III.3. Suppression de données

Syntaxe : Il s'agit de supprimer un ou plusieurs tuples d'une table. Pour ce faire, on écrit :

```
DELETE FROM nom_table
[WHERE condition] ;
```

Exemple : **DELETE FROM** ETUDIANT
WHERE NCE=1234 ;

Exemple :
Si on souhaite supprimer l'**employé** de **Employee_id** 5 de la table **Employee**, on écrit :

Employee	Employee_id	First_name	Last_name	Salary	Joining_date	Departement
	1	KONAN	Kinto	1000000	2019-01-20	Finance
	2	Jerry	Kansxo	6000000	2019-01-15	IT
	3	Philip	Jose	8900000	2019-02-05	Banking
	4	John	Abraham	2000000	2019-02-25	Insurance
	5	Michael	Mathew	2200000	2019-02-28	Finance
	6	Alex	chreketo	4000000	2019-05-10	IT
	7	Yohan	Soso	1230000	2019-06-20	Banking

```
DELETE FROM Employee
WHERE Employee_id= 7;
```

Employee	Employee_id	First_name	Last_name	Salary	Joining_date	Departement
	1	KONAN	Kinto	1000000	2019-01-20	Finance
	2	Jerry	Kansxo	6000000	2019-01-15	IT
	3	Philip	Jose	8900000	2019-02-05	Banking
	4	John	Abraham	2000000	2019-02-25	Insurance
	6	Alex	chreketo	4000000	2019-05-10	IT
	7	Yohan	Soso	1230000	2019-06-20	Banking

IV. Interrogation de données

IV.1. Généralités

Il s'agit de chercher un ou plusieurs tuples de la base de données.

Syntaxe :

L'ordre SELECT possède six clauses différentes, dont seules les deux premières sont obligatoires. Elles sont données ci-dessous dans l'ordre dans lequel elles doivent apparaître quand elles sont utilisées :

SELECT…
FROM…
[WHERE…
GROUP BY…
HAVING…
ORDER BY…**] ;**

IV.2. Projection

Tous les attributs d'une table :

```
SELECT *
FROM nom_table ;
```

Remarque : Il est possible de mettre le mot clé facultatif DISTINCT derrière l'ordre SELECT. Il permet d'éliminer les duplications : si, dans le résultat, plusieurs lignes sont identiques, une seule sera conservée.

Exemple :
L'exemple suivant renvoi tous les enregistrements de la table «**Employee**»:

```
SELECT *
FROM Employee;
```

Employee	Employee_id	First_name	Last_name	Salary	Joining_date	Departement
	1	KONAN	Kinto	1000000	2019-01-20	Finance
	2	Jerry	Kansxo	6000000	2019-01-15	IT
	3	Philip	Jose	8900000	2019-02-05	Banking
	4	John	Abraham	2000000	2019-02-25	Insurance
	6	Alex	chreketo	4000000	2019-05-10	IT
	7	Yohan	Soso	1230000	2019-06-20	Banking

Quelques attributs :

```
SELECT attribut1, attribut2, …
FROM nom_table ;
```

```
SELECT First_name, Departement
FROM Employee;
```

Employee	First_name	Departement
	KONAN	Finance
	Jerry	IT
	Philip	Banking
	John	Insurance
	Alex	IT
	Yohan	Banking

Exemple :
Liste des noms des **Departements** sans duplication

```
SELECT DISTINCT Departement
FROM Employee;
```

Employee	Departement
	Finance
	IT
	Banking
	Insurance

IV.3. Restriction

Les restrictions se traduisent en SQL à l'aide du prédicat « WHERE » comme suit :

```
SELECT attribut1, attribut2, …
FROM nom_table
WHERE predicat ;
```

Un prédicat simple est la comparaison de deux expressions ou plus au moyen d'un opérateur logique.
Les trois types d'expressions (arithmétiques, caractères ou dates) peuvent être comparées au moyen des opérateurs d'égalité ou d'ordre (=, !=, <, <=, >, >=) :
- pour les types date, la relation d'ordre est l'ordre chronologique
- pour les types caractères, la relation d'ordre est l'ordre lexicographique.

Nous résumons les principales formes de restrictions dans ce qui suit :

WHERE exp1 = exp2

Exemple :
Liste des étudiants qui s'appellent Ali
SELECT *

FROM ETUDIANT
WHERE PRENOM = 'Ali' ;

WHERE exp1 != exp2

Exemple : Liste des étudiants qui ne s'appellent pas Ali
SELECT *
FROM ETUDIANT
WHERE PRENOM != 'Ali' ;

WHERE exp1 > exp2
WHERE exp1 >= exp2

Exemple :
Récupérez tous les détails sur les employés qui ont adhéré après le 31 mars 2019

```
SELECT *
FROM employee
WHERE joining_date > '2019-03-31';
```

Employee	Employee_id	First_name	Last_name	Salary	Joining_date	Departement
	6	Alex	chreketo	4000000	2019-05-10	IT
	7	Yohan	Soso	1230000	2019-06-20	Banking

WHERE exp1 < exp2
WHERE exp1 <= exp2

Exemple :
Récupérez tous les détails sur les employés qui ont adhéré (Joining_date) avant le 1er mars 2019

```
SELECT *
FROM employee
WHERE joining_date < '2019-03-01';
```

Employee	Employee_id	First_name	Last_name	Salary	Joining_date	Departement
	1	KONAN	Kinto	1000000	2019-01-20	Finance
	2	Jerry	Kansxo	6000000	2019-01-15	IT
	3	Philip	Jose	8900000	2019-02-05	Banking
	4	John	Abraham	2000000	2019-02-25	Insurance

WHERE exp1 **BETWEEN** exp2 **AND** exp3
La condition est vrai si exp1 est compris entre exp2 et exp3 (bornes incluses)

Exemple : Liste des **Employes** ayant les **Salary** compris entre 5000000 et 10000000

```
SELECT *
FROM Employee
WHERE Salary BETWEEN 5000000 AND 10000000;
```

Employee	Employee_id	First_name	Last_name	Salary	Joining_date	Departement
	2	Jerry	Kansxo	6000000	2019-01-15	IT
	3	Philip	Jose	8900000	2019-02-05	Banking

WHERE exp1 **LIKE** exp2 :
LIKE teste l'égalité de deux chaînes en tenant compte des **caractères jokers** dans la 2ème chaîne :
«_» remplace un caractère exactement,
« % » remplace une chaîne de caractères de longueur quelconque (y compris de longueur nulle)

Exemple :
Liste des Employes ayant les noms commençant par K et contenant au moins 2 caractères.

```
SELECT *
FROM Employee
WHERE Last_name LIKE "K_%";
```

Employee	Employee_id	First_name	Last_name	Salary	Joining_date	Departement
	1	KONAN	Kinto	1000000	2019-01-20	Finance
	2	Jerry	Kansxo	6000000	2019-01-15	IT

WHERE exp1 **IN** (exp2, exp3, …)
Le prédicat est vrai si exp1 est égale à l'une des expressions de la liste entre parenthèses.

Exemple :
Liste des **Employee** ayant les **Last_name** appartenant à la liste ('chreketo', 'Jose', 'Brice')

```
SELECT *
FROM Employee
WHERE Last_name IN ('chreketo', 'Jose', 'Brice') ;
```

Employee	Employee_id	First_name	Last_name	Salary	Joining_date	Departement
	3	Philip	Jose	8900000	2019-02-05	Banking
	6	Alex	chreketo	4000000	2019-05-10	IT

WHERE exp **IS NULL**

Exemple :
Liste des étudiants dont les prénoms sont non définis

```
SELECT *
FROM Employee
WHERE First_name IS NULL ;
```

WHERE EXISTS (sous_interrogation)
La clause EXISTS est suivie d'une sous interrogation entre parenthèses, et prend la valeur **VRAI s'il existe au moins une** ligne satisfaisant les conditions de la sous interrogation.

Exemple :
Récupérez tous les détails d'un employé si ce dernier existe dans la table Reward? Ou autrement dit, trouver les employées ayant des primes.

```
SELECT e.*
FROM Employee AS e
WHERE EXISTS(
        SELECT r.Employee_ref_id
        FROM reward AS r
        WHERE e.Employee_id = r.Employee_ref_id
);
```

Employee	Employee_id	First_name	Last_name	Salary	Joining_date	Departement
	1	KONAN	Kinto	1000000	2019-01-20	Finance
	2	Jerry	Kansxo	6000000	2019-01-15	IT
	3	Philip	Jose	8900000	2019-02-05	Banking

```
SELECT e.*
FROM Employee AS e
WHERE EXISTS(
        SELECT r.Employee_ref_id
        FROM reward AS r
        WHERE e.Employee_id = r.Employee_ref_id
);
```

Employee	Employee_id	First_name	Last_name	Salary	Joining_date	Departement
	4	John	Abraham	2000000	2019-02-25	Insurance
	6	Alex	chreketo	4000000	2019-05-10	IT
	7	Yohan	Soso	1230000	2019-06-20	Banking

Exemple 2 : Liste des articles qui n'ont jamais été commandés.

```
SELECT id_article, designation
FROM article
WHERE NOT EXISTS (
                        SELECT id_article
                        FROM ligne
                        WHERE article.id_article = ligne.id_article);
```

Remarque :
On peut trouver, de même, les négations des prédicats BETWEEN, NULL, LIKE, IN, EXISTS à savoir : NOT BETWEEN, NOT NULL, NOT LIKE, NOT IN et NOT EXISTS.

IV.4. Tri

Les lignes constituant le résultat d'un SELECT sont obtenues dans un ordre indéterminé. La clause ORDER BY précise l'ordre dans lequel la liste des lignes sélectionnées sera donnée.

Syntaxe :

```
ORDER BY exp1 [DESC], exp2 [DESC], ...
```

L'option facultative DESC donne un tri par **ordre décroissant**. Par défaut, l'ordre est croissant.
Le tri se fait d'abord selon la première expression, puis les lignes ayant la même valeur pour la première expression sont triées selon la deuxième, ...
Remarque : Les valeurs nulles sont toujours en tête quel que soit l'ordre du tri (ascendant ou descendant).

Exemple :
Liste des **Employee** ordonnés par ordre croissant des **First_name**

```
SELECT *
FROM Employee
ORDER BY First_name;
```

Employee	Employee_id	First_name	Last_name	Salary	Joining_date	Departement
	6	Alex	chreketo	4000000	2019-05-10	IT
	2	Jerry	Kansxo	6000000	2019-01-15	IT
	4	John	Abraham	2000000	2019-02-25	Insurance
	1	KONAN	Kinto	1000000	2019-01-20	Finance
	3	Philip	Jose	8900000	2019-02-05	Banking
	7	Yohan	Soso	1230000	2019-06-20	Banking

Exemple :
Liste des **Employes** ordonnés par ordre croissant des **Departement** et décroissant des **Salary**

```
SELECT *
FROM Employee
ORDER BY Departement ASC, Salary DESC;
```

Employee	Employee_id	First_name	Last_name	Salary	Joining_date	Departement
	3	Philip	Jose	8900000	2019-02-05	Banking
	7	Yohan	Soso	1230000	2019-06-20	Banking
	1	KONAN	Kinto	1000000	2019-01-20	Finance
	4	John	Abraham	2000000	2019-02-25	Insurance
	2	Jerry	Kansxo	6000000	2019-01-15	IT
	6	Alex	chreketo	4000000	2019-05-10	IT

IV.5. Regroupement

IV.5.1. La clause GROUP BY

Il est possible de subdiviser une table en groupes, chaque groupe étant l'ensemble de lignes ayant une valeur commune.

Syntaxe :

```
GROUP BY exp1, exp2, ...
```

Cette clause **groupe en une seule ligne** toutes les lignes pour lesquelles exp1, exp2,... ont la même valeur.

Remarques :

- Cette clause se place juste après la clause WHERE, ou après la clause FROM si la clause WHERE n'existe pas.
- Des lignes peuvent être éliminées avant que le groupe ne soit formé grâce à la clause WHERE.

Exemples :
Récupérez le **département** et le **salaire total** des employés, regroupés par département.

```
SELECT Departement, sum(Salary) AS total
FROM employee
GROUP BY Departement;
```

Departement	total
Banking	10130000
Finance	1000000
Insurance	2000000
IT	10000000

Exemples :
Liste des départements ainsi que le nombre de leurs employés

```
SELECT Departement, COUNT(*)
FROM Employee
GROUP BY Departement;
```

Departement	COUNT(*)
Banking	2
Finance	1
Insurance	1
IT	2

Exemples :
Récupérez le nombre d'employés regroupé par l'année et le mois d'adhésion.

```
SELECT YEAR(joining_date) AS "Année",  MONTH(joining_date) AS "Mois",
      count(*) AS total_emp
FROM employee
GROUP BY YEAR(joining_date), MONTH(joining_date);
```

Année d'adhésion	Mois d'adhésion	total_emp
2019	1	2
2019	2	2
2019	5	1
2019	6	1

Exemples :
Liste des articles (id_article, designation) ainsi que le nombre de fois que chacun a été commandé.

SELECT ligne.id_article, designation, **COUNT(ligne.**id_article**)**
FROM ligne, article
where ligne.id_article = article.id_article
GROUP BY id_article;

Liste des départements ainsi que le nombre de leurs secrétaires

SELECT DEPT, **COUNT(*)**
FROM EMP
WHERE POSTE = 'SECRETAIRE'
GROUP BY DEPT;

IV.5.2. La clause HAVING

HAVING sert à préciser quels groupes doivent être sélectionnés.
Elle se place après la clause GROUP BY.
Syntaxe :

```
HAVING predicat
```

Remarque : Le prédicat suit la même syntaxe que celui de la clause WHERE. Cependant, il ne peut porter que sur des caractéristiques de groupe (fonctions de groupe (MIN(), MAX(), AVG(), COUNT()) ou expression figurant dans la clause GROUP BY)

Exemple :

Employee	Employee_id	First_name	Last_name	Salary	Joining_date	Departement
	1	KONAN	Kinto	1000000	2019-01-20	Finance
	2	Jerry	Kansxo	6000000	2019-01-15	IT
	3	Philip	Jose	8900000	2019-02-05	Banking
	4	John	Abraham	2000000	2019-02-25	Insurance
	6	Alex	chreketo	4000000	2019-05-10	IT
	7	Yohan	Soso	1230000	2019-06-20	Banking

```
SELECT Departement, COUNT(*)
FROM Employee
GROUP BY Departement
HAVING COUNT(*) > 1 ;
```

Departement	COUNT(*)
Banking	2
IT	2

Exemples :
Liste des articles (id_article, designation) qui ont été commandé au moins deux fois. Ainsi que le nombre de fois qu'ils ont été commandé.

```
SELECT ligne.id_article, designation, COUNT(ligne.id_article)
FROM ligne, article
WHERE ligne.id_article = article.id_article
GROUP BY id_article
HAVING COUNT(ligne.id_article) >= 2 ;
```

IV.6. Opérateurs ensemblistes

IV.6.1. Union

L'opérateur UNION permet de fusionner deux sélections de tables pour obtenir un ensemble de lignes égal à la réunion des lignes des deux sélections. Les lignes communes n'apparaîtront qu'une fois.

Exemple : Liste des ingénieurs des deux filiales

SELECT * FROM EMP1 **WHERE** POSTE = 'INGENIEUR'
UNION
SELECT * FROM EMP2 **WHERE** POSTE = 'INGENIEUR' **;**

IV.6.2. Différence

L'opérateur MINUS permet d'ôter d'une sélection les lignes obtenues dans une deuxième sélection.

Exemple : Liste des départements qui ont des employés dans la première filiale mais pas dans la deuxième

SELECT DEPT **FROM** EMP1
MINUS
SELECT DEPT **FROM** EMP2**;**

IV.6.3. Intersection

L'opérateur INTERSECT permet d'obtenir l'ensemble des lignes communes à deux interrogations.

Exemple : Liste des départements qui ont des employés dans les deux filiales

SELECT DEPT **FROM** EMP1
INTERSECT
SELECT DEPT **FROM** EMP2**;**

IV.7. Jointure

IV.7.1. Définition

Quand on précise plusieurs tables dans la clause **FROM**, on obtient le **produit cartésien** des tables.
Le produit cartésien de deux tables offre en général peu d'intérêt.
Ce qui est normalement souhaité, c'est de joindre les informations de diverses tables, en précisant quelles relations les relient entre elles. C'est la clause WHERE qui permet d'obtenir ce résultat. Elle vient limiter cette sélection en ne conservant que le **sous-ensemble** du produit cartésien qui satisfait le prédicat.

Exemple : Liste des noms des étudiants avec les noms de leurs classes

SELECT NOMETUDIANT, NOMCLASSE
FROM ETUDIANT, CLASSE
WHERE ETUDIANT.NUMCLASSE = CLASSE.NUMCLASSE ;

Exemple : Liste des articles commandés avec leurs désignations

SELECT article.id_article, article.designation
FROM article, ligne

WHERE article.id_article = ligne.id_article ;

Exemple
Récupérer le prénom (First_name), le montant de la récompense (amount) pour les employés qui ont des récompenses.

Employee	Employee_id	First_name	Last_name	Salary	Joining_date	Departement
	1	KONAN	Kinto	1000000	2019-01-20	Finance
	2	Jerry	Kansxo	6000000	2019-01-15	IT
	3	Philip	Jose	8900000	2019-02-05	Banking
	4	John	Abraham	2000000	2019-02-25	Insurance
	5	Michael	Mathew	2200000	2019-02-28	Finance
	6	Alex	chreketo	4000000	2019-05-10	IT
	7	Yohan	Soso	1230000	2019-06-20	Banking

reward	Employee_ref_id	date_reward	amount
	1	2019-05-11	1000
	2	2019-02-15	5000
	3	2019-04-22	2000
	1	2019-06-20	8000

```
SELECT First_name, amount
FROM Employee E
INNER JOIN Reward R
ON E.employee_id = R.employee_ref_id;
```

OU

```
SELECT First_name, amount
FROM Employee E, Reward R
WHERE E.employee_id = R.employee_ref_id;
```

First_name	amount
KONAN	1000
Jerry	5000
Philip	2000
KONAN	8000

Exemple
Récupérer le prénom, le montant de la récompense pour les employés qui ont des récompenses avec un montant supérieur à 2000.

```
SELECT First_name, amount
FROM Employee E, Reward R
WHERE E.employee_id = R.employee_ref_id AND amount > 2000;
```

First_name	amount
Jerry	5000
KONAN	8000

```
SELECT First_name, amount
FROM Employee E , Reward R
WHERE E.employee_id = R.employee_ref_id AND amount > 2000;
```

IV.7.2. Jointure d'une table à elle même

Il peut être utile de rassembler des informations venant d'une ligne d'une table avec des informations venant d'une autre ligne de la même table.
Dans ce cas, il faut renommer au moins l'une des deux tables en lui donnant un synonyme, afin de pouvoir préfixer sans ambiguïté chaque nom de colonne.

Considerons la table suivante :

E	Employee_id	First_name	Last_name	Salary	Joining_date	Departement	id_s
	1	KONAN	Kinto	1000000	2019-01-20	Finance	7
	2	Jerry	Kansxo	6000000	2018-01-15	IT	4
	3	Philip	Jose	8900000	2019-02-05	Banking	7
	4	John	Abraham	2000000	2018-02-25	Insurance	
	6	Alex	chreketo	4000000	2019-05-10	IT	7
	7	Yohan	Soso	1230000	2019-06-20	Banking	4

S	Employee_id	First_name	Last_name	Salary	Joining_date	Departement	id_s
	1	KONAN	Kinto	1000000	2019-01-20	Finance	7
	2	Jerry	Kansxo	6000000	2018-01-15	IT	4
	3	Philip	Jose	8900000	2019-02-05	Banking	7
	4	John	Abraham	2000000	2018-02-25	Insurance	
	6	Alex	chreketo	4000000	2019-05-10	IT	7
	7	Yohan	Soso	1230000	2019-06-20	Banking	4

Exemple : Lister les employés qui ont un supérieur en indiquant pour chacun le nom de son supérieur

```
SELECT E.First_name EMPLOYE, S.First_name SUPERIEUR
FROM emp E, emp S
WHERE E.Employee_id = S.id_s ;
```

V. Contrôle de données

V.1. Gestion des utilisateurs

Tout accès à la base de données s'effectue par l'intermédiaire de la notion d'utilisateur (compte Oracle). Chaque utilisateur est défini par :

- un nom d'utilisateur
- un mot de passe
- un ensemble de privilèges

V.1.1. Création d'un utilisateur

Syntaxe : Pour créer un utilisateur, on doit spécifier le nom de l'utilisateur ainsi que le mot de passe via L'instruction :

CREATE USER utilisateur **IDENTIFIED BY** mot_de_passe ;

Exemple :

CREATE USER KOFFI IDENTIFIED BY "Ae3OPd"

V.1.2. Modification d'un compte utilisateur

Syntaxe : Pour modifier le mot de passe d'un utilisateur, on écrit :

ALTER USER utilisateur **IDENTIFIED BY** nouveau_mot_de_passe ;

Exemple :

ALTER USER ALI **IDENTIFIED BY** A23ePs ;

V.1.3. Suppression d'un utilisateur

Syntaxe : Pour supprimer un compte utilisateur, on écrit :

DROP USER utilisateur [**CASCADE**] ;

L'utilisation de CASCADE signifie que la suppression de l'utilisateur est accompagnée par la suppression de tous les schémas qu'il a créés.

Exemple :

DROP USER Ali **CASCADE** ;

V.2. Gestion des privilèges

V.2.1. Attribution de privilèges

Un privilège peut être attribué à un utilisateur par l'ordre GRANT.
Syntaxe :

GRANT privilège
[**ON** table]
TO utilisateur [**WITH GRANT OPTION**] ;

Remarque : Des droits peuvent être accordés à tous les utilisateurs par un seul ordre GRANT en utilisant le mot réservé PUBLIC à la place du nom d'utilisateur.
Principaux **Privilèges** :
SELECT : lecture
INSERT : insertion
UPDATE : mise à jour
DELETE : suppression
DBA, ALL : tous les privilèges

Si la clause WITH GRANT OPTION est spécifiée, le bénéficiaire peut à son tour assigner le privilège qu'il a reçu à d'autres utilisateurs.

Exemples :

```
GRANT SELECT
ON employee
TO PUBLIC ;
```

```
GRANT UPDATE, DELETE
ON employee
TO KOFFI WITH GRANT OPTION ;
```

V.2.2. Suppression des privilèges

Un privilège peut être enlevé à un utilisateur par l'ordre REVOKE.
Syntaxe :

```
REVOKE privilège
[ON table]
FROM utilisateur ;
```

Exemples :

```
REVOKE SELECT
ON ETUDIANT
FROM Ali ;
```