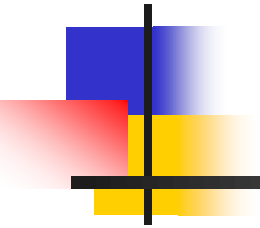
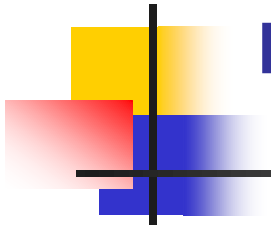


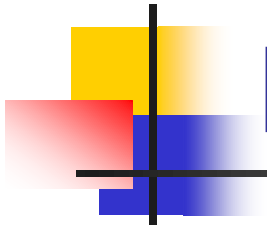
Bases de données relationnelles

Hicham BEHJA



Plan du cours

- **Rappel sur les SGBD**
- **Architecture d'un SGBD**
- **Algèbre Relationnelle**
- **Normalisation**

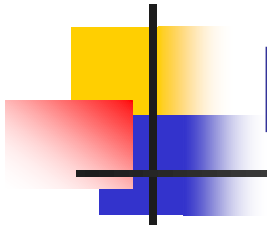


Bases de données

Une base de données est ensemble de données :

- représentant une partie du monde réel
- stockées en mémoire secondaire
- pouvant être interrogées et mises à jour
- servant de support à une ou plusieurs applications

Mais… base de données ≠ banque de données



Bases de données

Une collection de données est appelée **base de données** si:

- ***format connu*** (défini au niveau du système et non uniquement au niveau des programmes utilisant les données) et défini par des ***méta-données*** (données décrivant des données)
- Données stockées, recherchées, modifiées uniquement par un type spécial de programmes appelé ***système de gestion de bases de données***
- Données manipulées sous le contrôle de ***transactions*** (ensemble de règles formelles assurant l'intégrité des données

Système de gestion de bases de données (SGBD)

Un SGBD est un ensemble de logiciels-système permettant aux utilisateurs de:

- mettre en forme
- sauvegarder,
- mettre à jour (modifier, insérer, supprimer)
- rechercher

efficacement des données spécifique dans une très grande masse d'informations partagées entre plusieurs utilisateurs

Fonctionnalités d'un SGBD (1)

- **Contrôler la redondance d'informations**
- **Partage des données**
- Fiabilité des données
  - Cohérence et contraintes d'intégrité
  - Sûreté du fonctionnement
    - Notion de transaction atomique
    - Techniques de sauvegarde et/ou de journalisation
    - Procédures de reprise sur panne

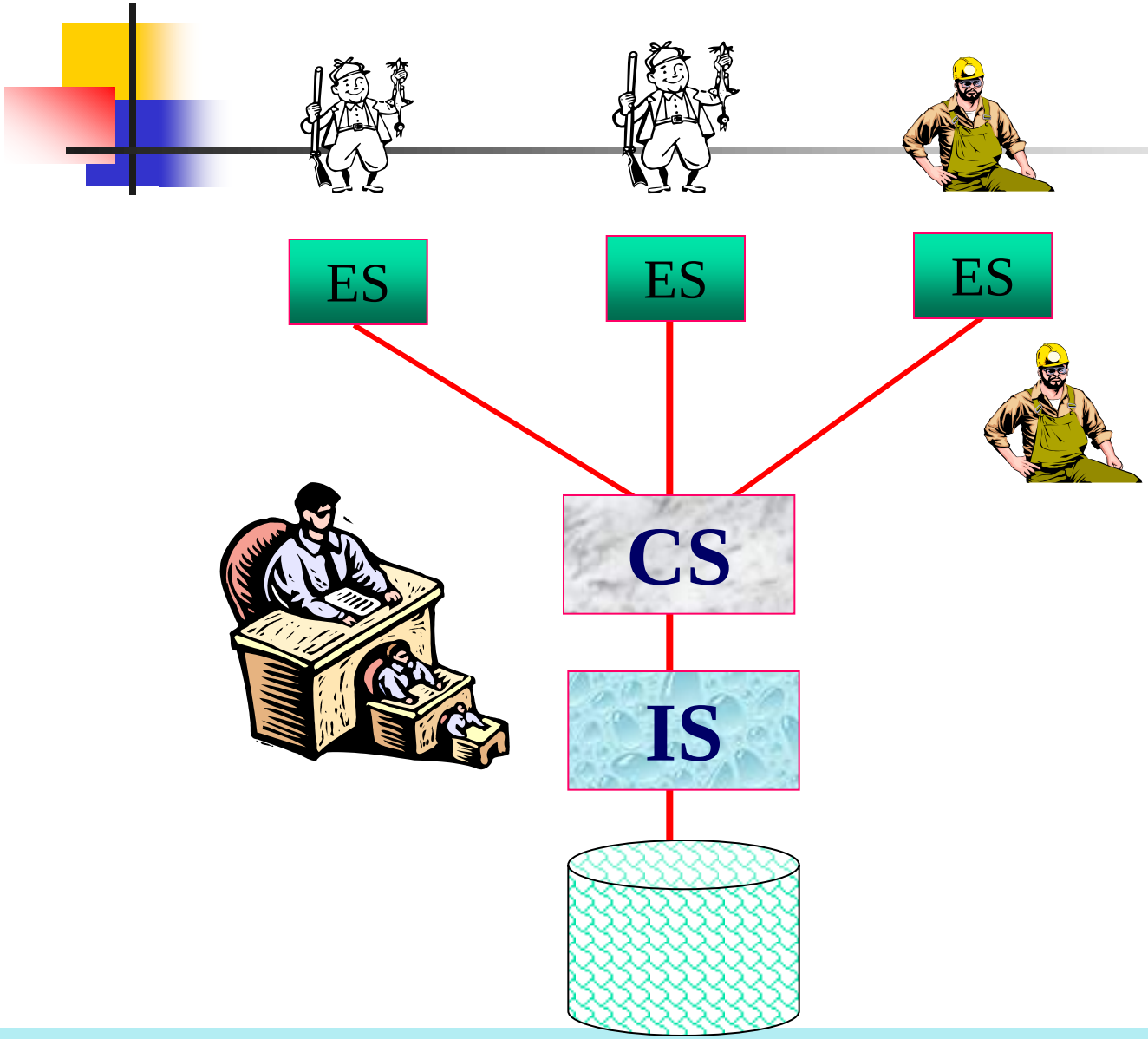
Fonctionnalités d'un SGBD (2)

- Sécurité d'accès
  - commandes d'autorisation
- Partage et accès concurrents
  - techniques de verrouillage
- Interrogation : langages de requêtes
  - déclaratifs et incomplets

  → intégration du langage de requête et d'un langage de programmation

  → dysfonctionnements (typage et traitement ensembliste)

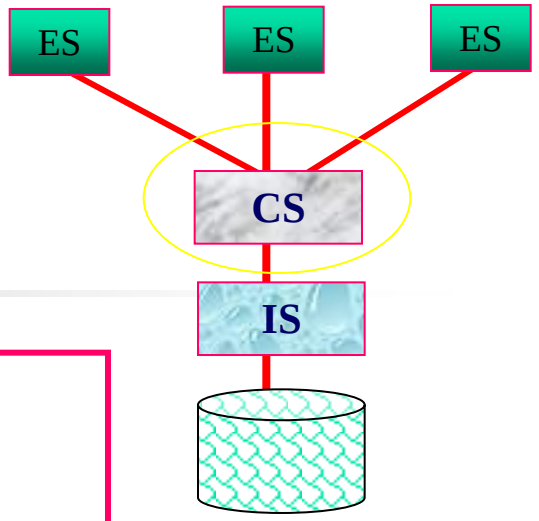
Architecture ANSI-SPARC



Architecture ANSI-SPARC

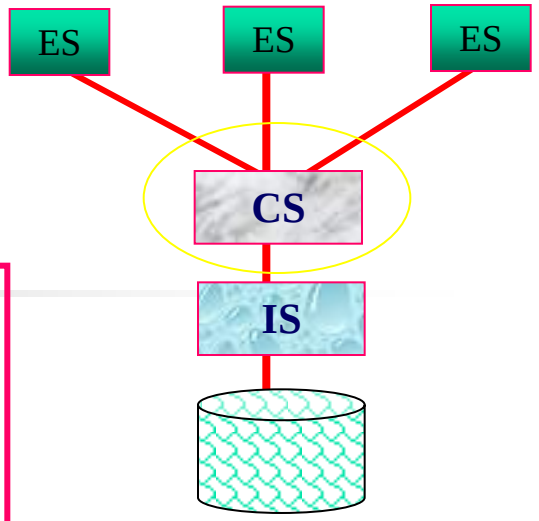
- Un standard pour tout SGBD
- Proposé vers 1965 par Charles Bachman
  - Auteur du concept de la BD
  - Créateur du modèle de données réseau (Codasyl)
  - Plus tard de l'architecture Open System Interconnection (OSI)
  - Prix Turing
    - La plus haute récompense scientifique en informatique en USA

Architecture ANSI-SPARC
Schéma Conceptuel (CS)



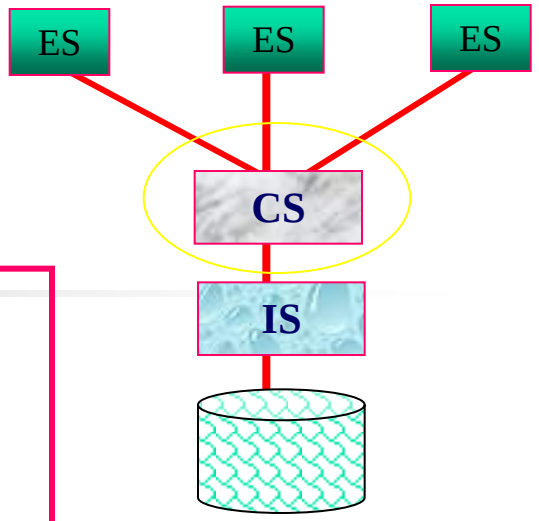
- D'une manière abstraite: un *modèle* conceptuel de l'univers réel de la BD
- D'une manière appliquée : la définition logique de la BD
  - Une et une seule
  - Les données logiques, leurs structures et types
    - Relations, attributs, domains
    - Entités…
    - Objets, Types, Classes
  - Leur manipulations
    - procédures, fonctions, méthodes…

Architecture ANSI-SPARC
Schéma Conceptuel (CS)



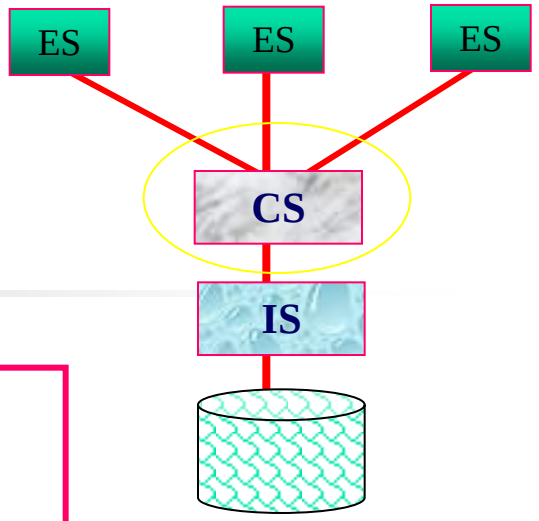
- La définition logique de la BD (suite)
  - Les liens sémantiques
    - Données d'un même objet réel
  - Les contraintes d'intégrité
    - Mono-valeur
    - Référentielles
    - Variées
  - Les contraintes de sécurité
    - Qui peut manipuler quoi

Architecture ANSI-SPARC
Schéma Conceptuel (CS)



- Le CS est définit par le DBA seul
  - Complexité
  - Sécurité
- Par l'intégration de données des différentes applications de la BD
  - Plusieurs méthodes de conception, plus ou moins formelles
    - Entité-Relations
    - Objets & Fonctions
    - Merise
    - Normalisation relationnelle

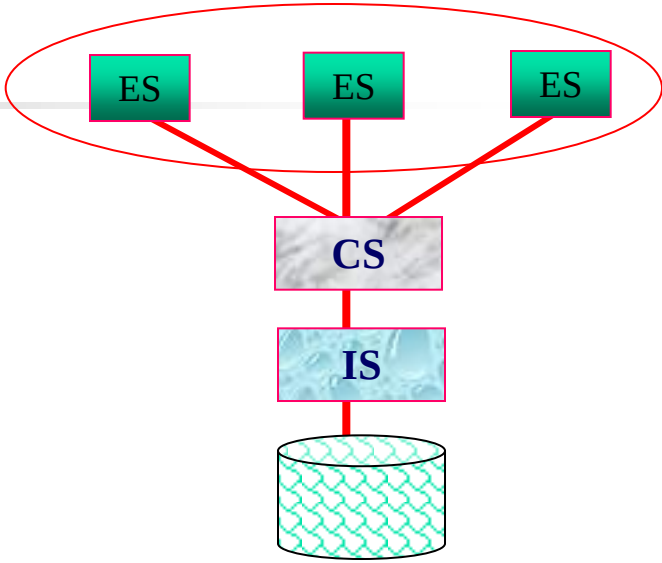
Architecture ANSI-SPARC
Schéma Conceptuel (CS)



- La BD (donc le CS) est définit en utilisant :
  - Le *langage de définition de données*
  - La BD est manipulée au niveau de CS à travers:
  - Le *langage de manipulation de données*
- Les deux sous-langages forment:
  - Le *langage de base de données*
    - En général incomplet au sens de la machine de Turing
    - SQL pour une BD relationnelle

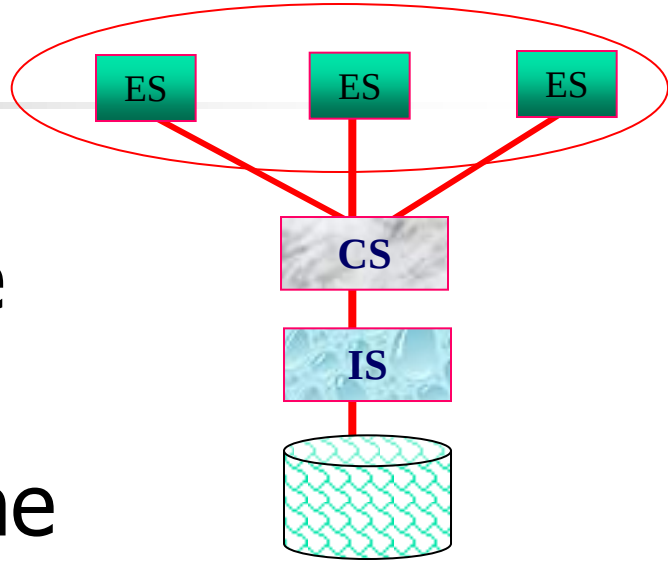
Architecture ANSI-SPARC

Schémas Externes (ESs)



- Un ES = un sous-schéma d'une BD
- Dérivé du CS
  - La dérivation est définie dans le ES
- Définit une *vue* de la BD
  - Une sous-base virtuelle
    - En général partielle
  - Pour des usagers ad-hoc
    - Vues 4-GL, orientés Web notamment (HTML, XML…)
  - Pour des programmeurs d'application
    - Vues SQL, pour SQL imbriqué ou Vbasic, ou procédures stockées…

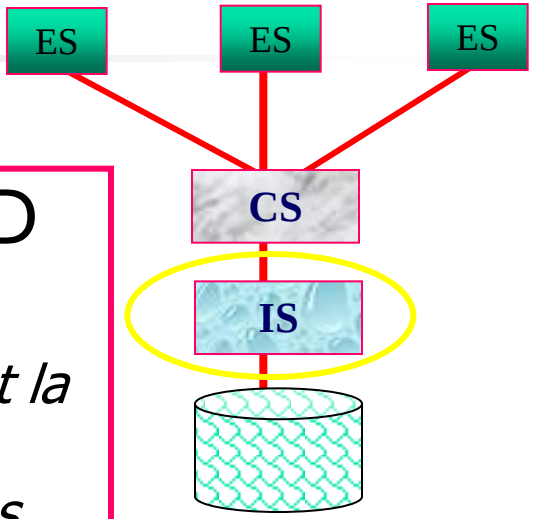
Architecture ANSI-SPARC
Schémas Externes (ESs)



- Une BD est en général munie de plusieurs différentes ESs
- Mais tous ont le CS comme racine commune
  - Donc tous doivent être accepté par le DBA
    - Perte d'autonomie de l'usager

Architecture ANSI-SPARC

Schéma Interne (IS)



- Définit la représentation interne de la BD
  - Niveau interne ou *physique*
    - *Les disques, fichiers hachés, arbres-B… contenant la BD*
    - *La représentation physique de valeurs de données*
      - *Réel, entier, texte, OLE…*
      - *Encodage…*
- Définit l'application du CS sur le IS
  - Selon le principe de l'indépendance de niveaux logique et physique

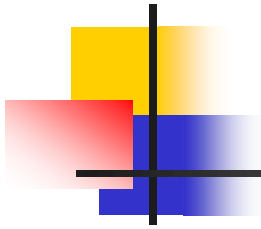
Le modèle relationnel de données

- **Défini en 1970 par Codd**
- **Simple facile à appréhender, même pour un non spécialiste**
- **Repose sur de solides bases théoriques**
- **Le modèle relationnel représente l'information dans une collection de relations.**

Le modèle relationnel de données (Définitions)

- un **domaine** D est un ensemble de valeurs atomiques. Le terme atomique signifie que ces valeurs sont considérées comme insécables au niveau du modèle.
- un schéma de **relation** R, dénoté par R(A1, A2, ..., An), est un ensemble d'attributs R = {A1, A2, ..., An}. A chaque attribut Ai est associé un domaine Di, Ai indiquant le rôle joué par le domaine Di dans la relation R. Un schéma de relation décrit une relation et représente l'intension de celle-ci.
- Le degré de la relation est le nombre d'attributs de celle-ci.

- Exemple :
- ETUDIANT(Nom, CNE, adresse, age, diplôme)



Définition d'une BD relationnelle

La définition d'une BD suppose l'existence de:

- un ensemble d'attributs
- un ensemble de domaines (ou types, ou valeurs)
- un ensemble de noms de tables

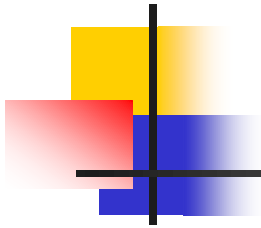


Schéma de relation

Définition: *schéma de relation R* =
ensemble fini de la forme

$$R = \{A_1:dom(A_1),\ldots, A_n: dom(A_n)\}$$

où A_i est un attribut et $dom(A_i)$ le domaine de A_i

noté $R = \{A_1,\ldots, A_n\}$

N-uplets et relations

Définition: un ***n-uplet t sur R*** est une fonction qui associe à chaque attribut de A_i de R une valeur de son domaine dom(A_i)

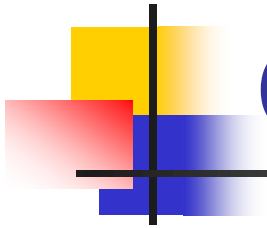
- Si $t(A_i) = a_i$ pour tout i, alors t est représenté par $a_i ... a_n$

Définition: une ***relation r sur R*** est un ensemble de n-uplets sur R

- R est appelé schéma de r, et noté sch(r)

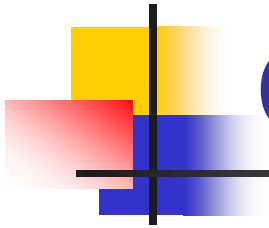
Caractéristiques des relations

- une relation est un ensemble de nuplets, il n'y a donc pas de notion d'ordre sur les nuplets,
- par contre un nuplet est un séquence ordonnée d'attributs,
- une valeur d'attribut est atomique mais peut être éventuellement nulle (valeur particulière qui indique que la valeur est manquante).



Contraintes d'intégrité

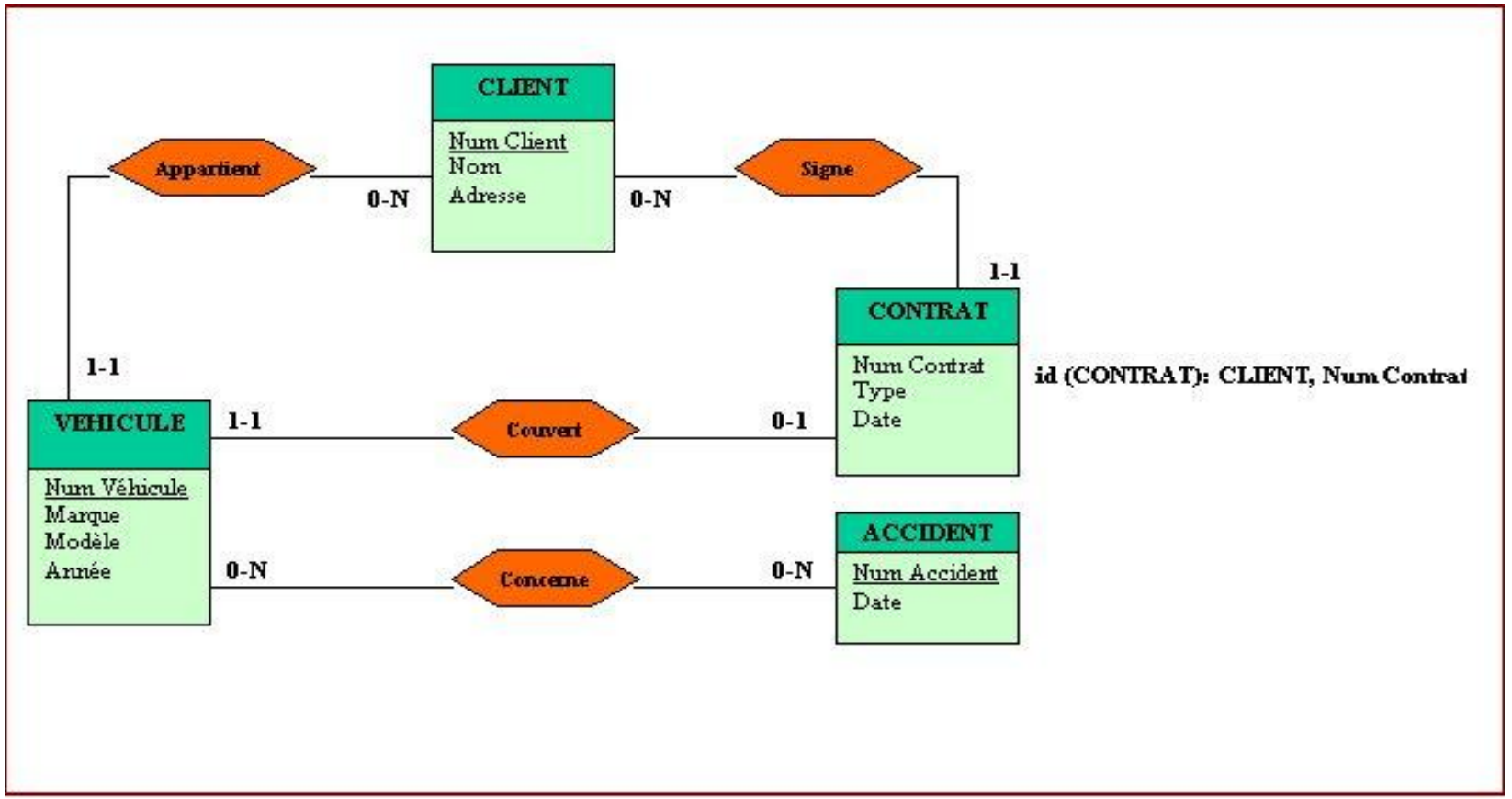
- **Elles permettent d'assurer la cohérence des données. Les contraintes d'intégrité sont :**
  - Contrainte de domaine : restriction de l'ensemble des valeurs possibles d'un attribut.
  - Contrainte de clé : définit un sous-ensemble minimal des colonnes tel que la table ne puisse contenir deux lignes ayant mêmes valeurs pour ces colonnes.
    - Il existe trois types de clés:
      Clé primaire : Ensemble minimum d'attributs qui permet de distinguer chaque n-uplet de la table par rapport à tous les autres.Chaque table doit avoir une clé primaire.
    - Clé candidate : Ensemble minimum d'attributs susceptibles de jouer le rôle de la clé primaire.
    - Clé étrangère : fait référence à la clé primaire d'une autre table.



Contraintes d'intégrité

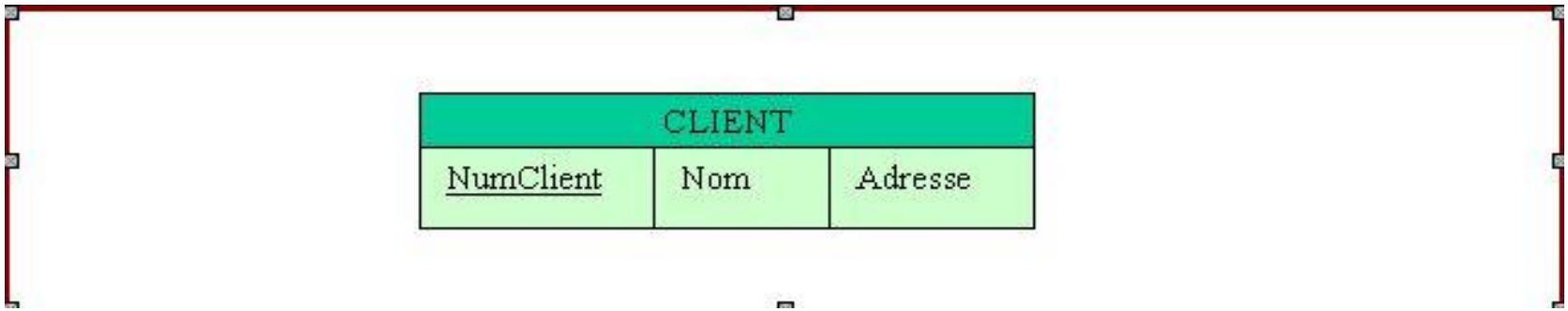
- Contrainte obligatoire : précise qu'un attribut ou plusieurs attributs doivent toujours avoir une valeur.
- Contrainte d'intégrité référentielle ou d'inclusion: lie deux colonnes ou deux ensembles de colonnes de deux tables différentes.

Règles à suivre pour concevoir un schéma relationnel



Règles à suivre pour concevoir un schéma relationnel

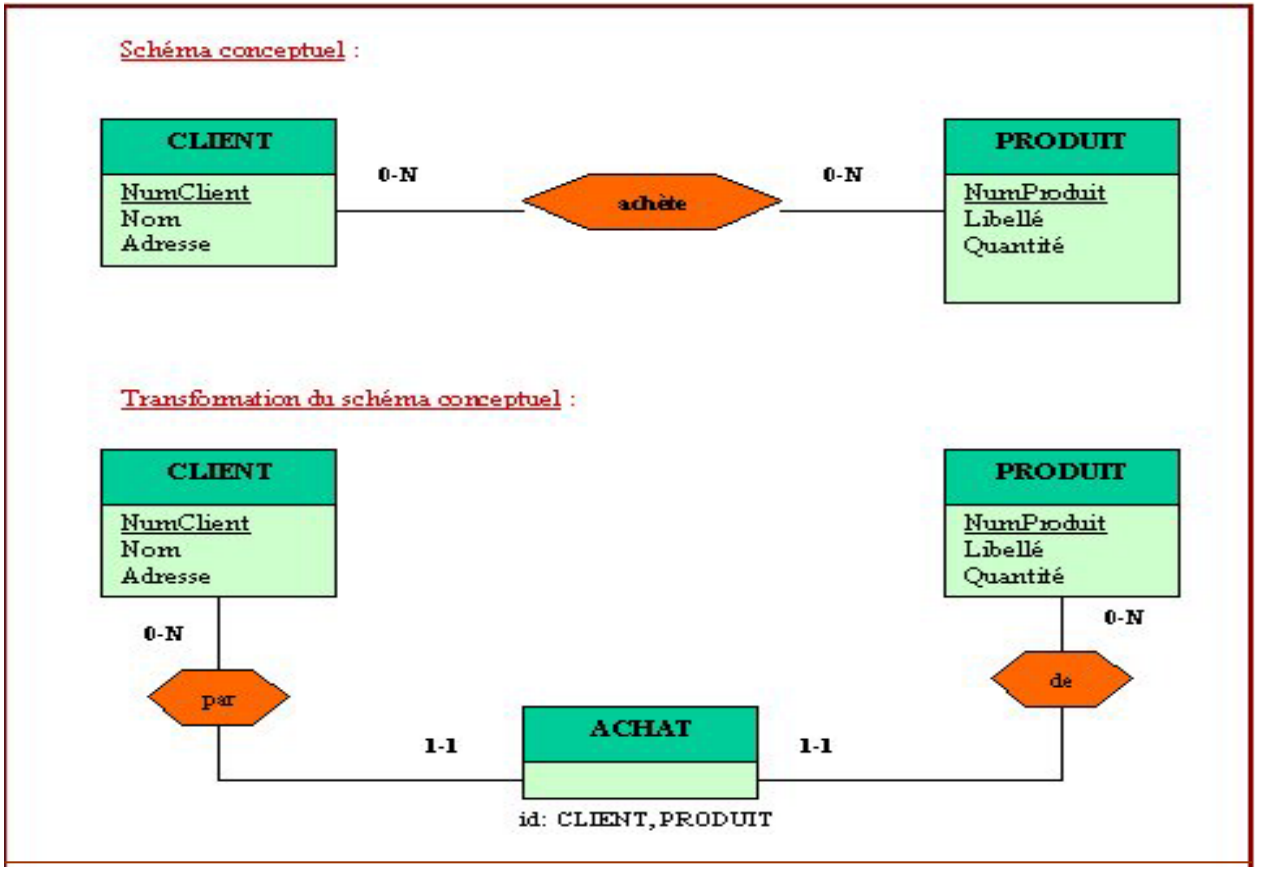
- **Règle I** : Toute entité est traduite en une table relationnelle dont les caractéristiques sont les suivantes :
  - le nom de la table est le nom de l'entité ;
  - la clé de la table est l'identifiant de l'entité ;
  - les autres attributs de la table forment les autres colonnes de la table.



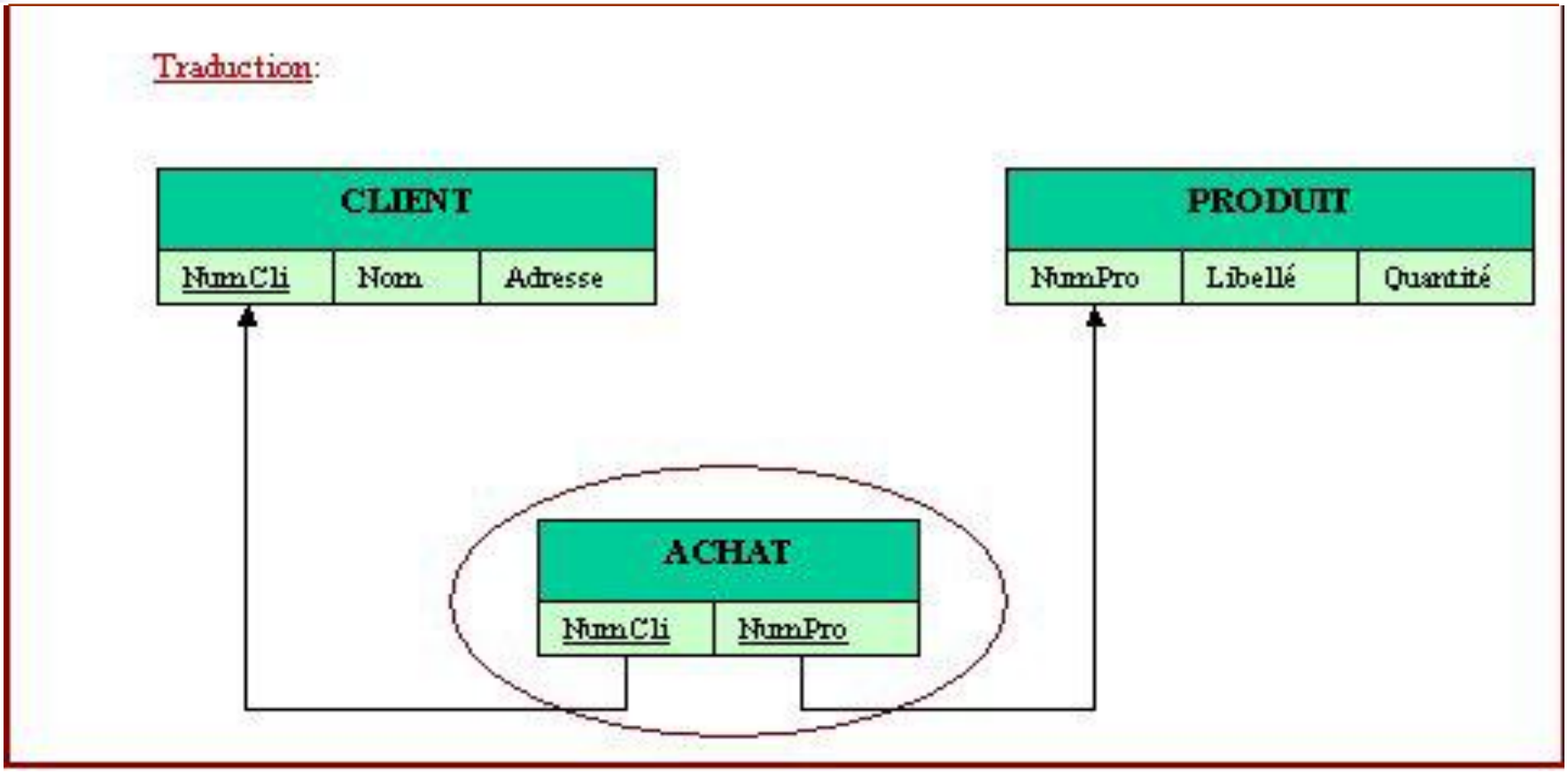
Règles à suivre pour concevoir un schéma relationnel

- **Règle II** : Toute relation binaire plusieurs à plusieurs est traduite en une table relationnelle dont les caractéristiques sont les suivantes :
  - le nom de la table est le nom de la relation ;
  - la clé de la table est formée par la concaténation des identifiants des entités participant à la relation ;
  - les attributs spécifiques de la relation forment les autres colonnes de la table.
- Une contrainte d'intégrité référentielle est générée entre chaque colonne clé de la nouvelle table et la table d'origine de cette clé.

Règles à suivre pour concevoir un schéma relationnel



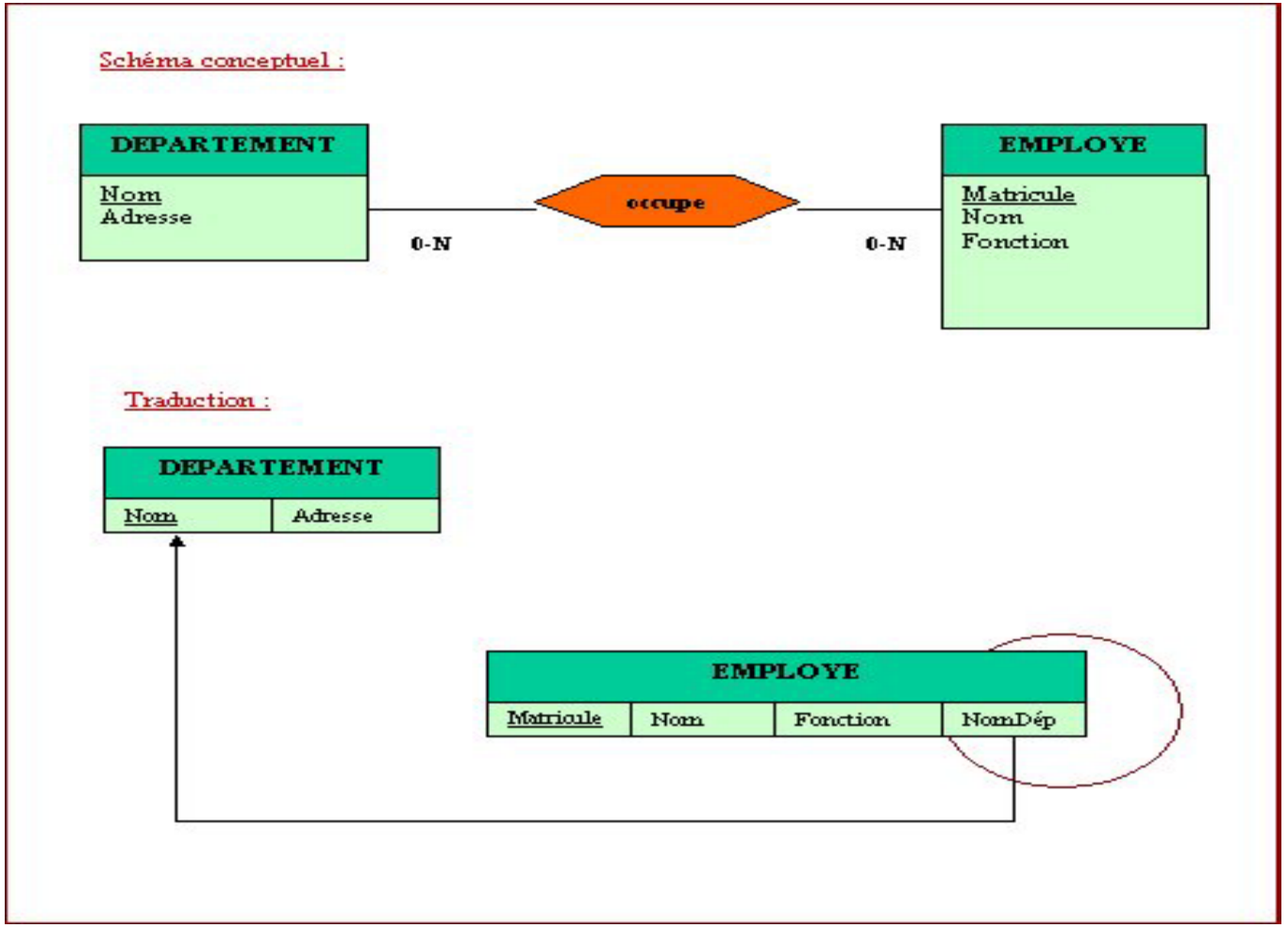
Règles à suivre pour concevoir un schéma relationnel



Règles à suivre pour concevoir un schéma relationnel

- **Règle III** : Toute relation binaire un à plusieurs est traduite :
  1. soit par un report de clé : l'identifiant de l'entité participant à la relation côté N est ajoutée comme colonne supplémentaire à la table représentant l'autre entité. Cette colonne est parfois appelée *clé étrangère*. Le cas échéant, les attributs spécifiques à la relation sont eux aussi ajoutés à la même table ;
  2. soit par une table spécifique dont les caractéristiques sont les suivantes :
     - le nom de la table est le nom de la relation ;
     - la clé de la table est l'identifiant de l'entité participent à la relation côté 1 ;
     - les attributs spécifiques de la relation forment les autres colonnes de la table.

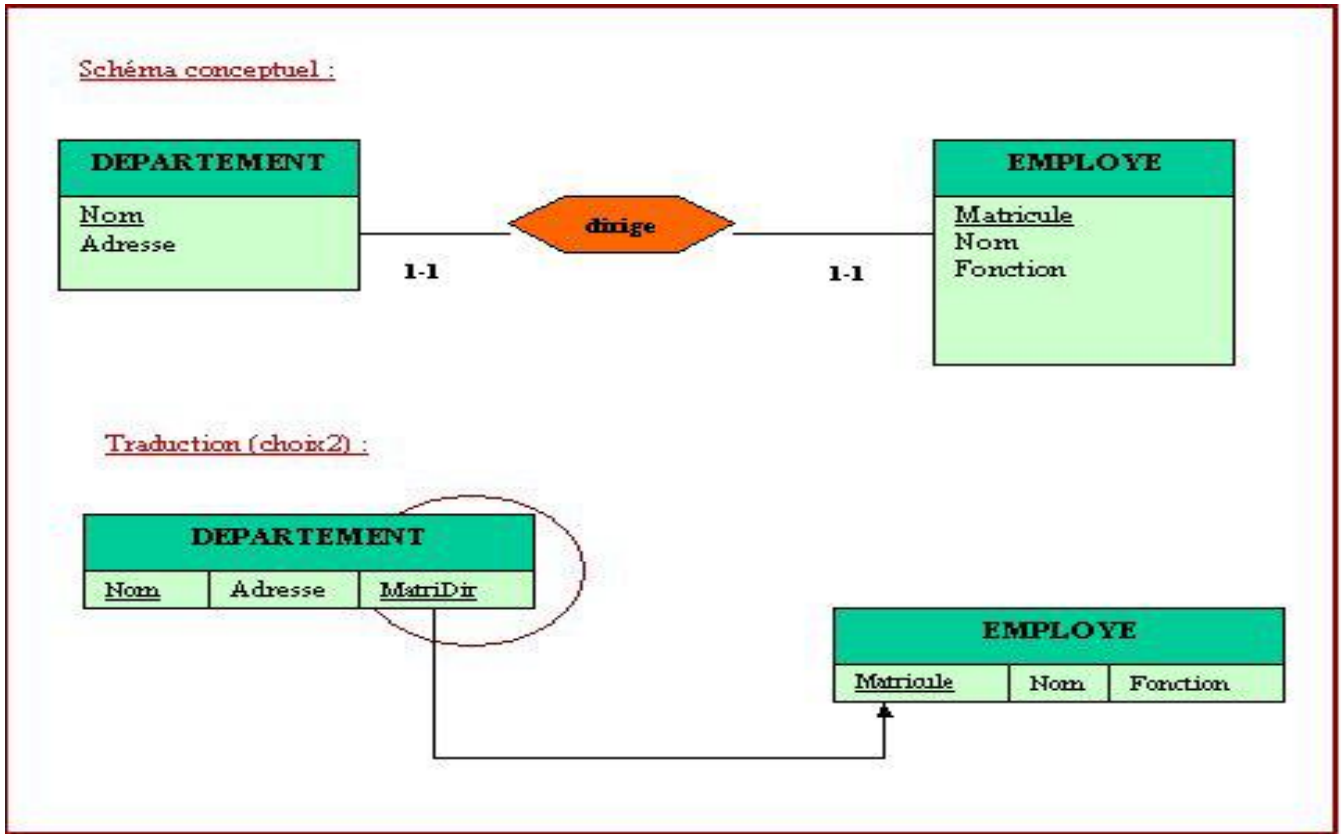
Règles à suivre pour concevoir un schéma relationnel



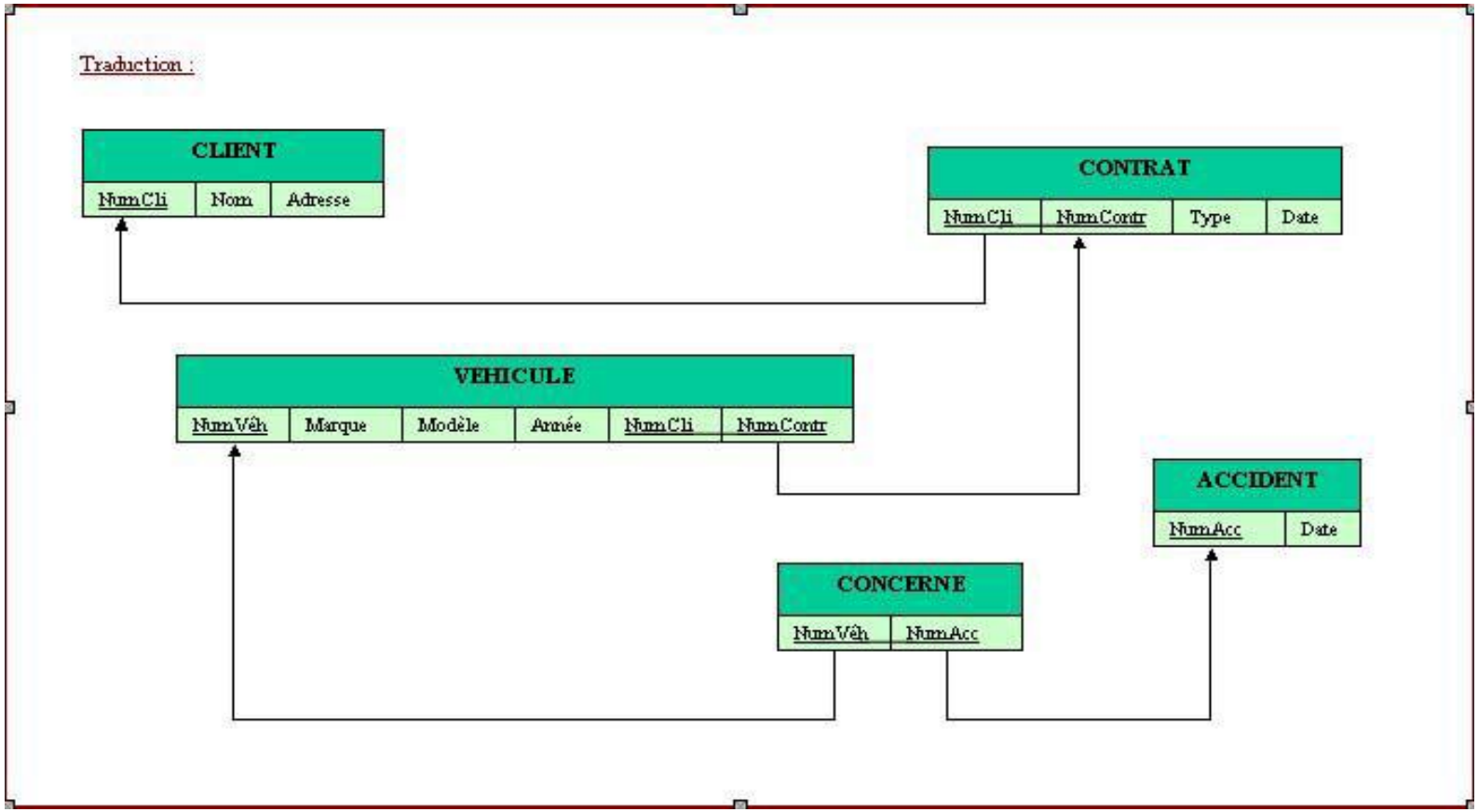
Règles à suivre pour concevoir un schéma relationnel

- **Règle IV** : Toute relation binaire un à un est traduite, au choix, par l'une des trois solutions suivantes :
  - fusion des tables des entités qu'elle relie (choix1) ;
  - report de clé d'une table dans l'autre (choix2) ;
  - création d'une table spécifique reliant les clés des deux entités (choix3).
  - Les attributs spécifiques de cette relation sont ajoutés à la table résultant de la fusion (choix1), reportés avec la clé (choix2), ou insérés dans la table spécifique (choix3).

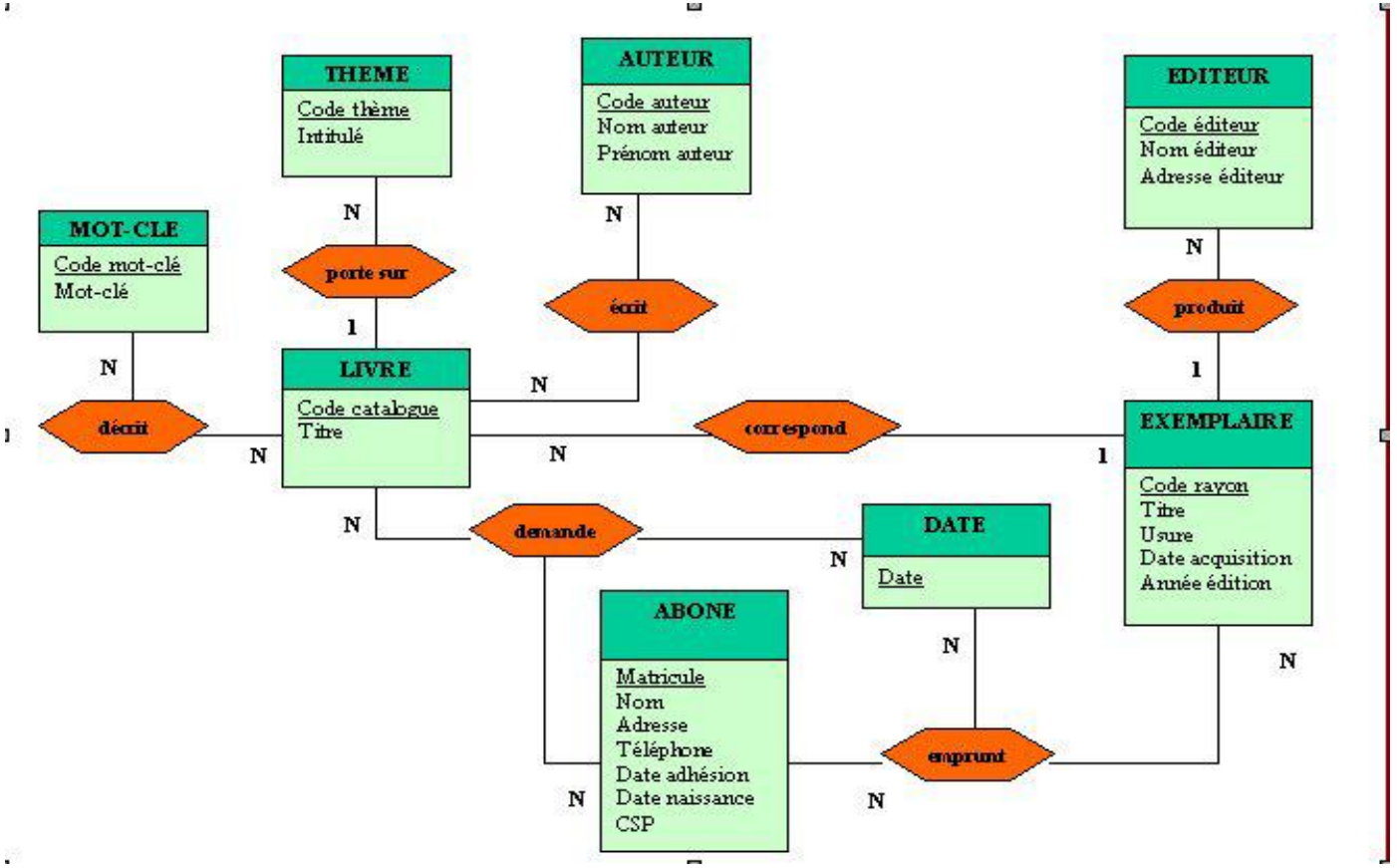
Règles à suivre pour concevoir un schéma relationnel



Règles à suivre pour concevoir un schéma relationnel



Règles à suivre pour concevoir un schéma relationnel



Normalisation des relations

- Objectif : Définir un bon schéma relationnel qui décrit bien l'entreprise.

- Exemple : on veut décrire les produits et leurs fournisseurs. On peut le faire avec le schéma suivant, schéma 1 (on suppose que chaque produit est d'une couleur unique) :

  Produit (NP, NomP, Couleur, Poids)
  Fournisseur (NF, NomF, Adr, Tel)
  Livraison (NP, NF, Date, Qté)

- Autre schéma proposé pour le même sujet, schéma 2 :

  Produit (NP, NomP, Couleur, Poids)
  Fournisseur (NF, NomF, Adr)
  Livraisonbis (NP, NF, Date, TélF, Qté)

Normalisation des relations

- Exemples de problèmes rencontrés lors des mises à jour de la base de données décrite par le schéma 2 :
  - S'il n'y a plus de livraison pour un fournisseur son numéro de téléphone est perdu.
  - S'il existe N livraisons pour un fournisseur, le numéro TélF est répété N fois, il faut vérifier que c'est le même.
  - Pour insérer une nouvelle livraison, il faut enregistrer à nouveau ce numéro TelF.

⇒ Ces problèmes n'existent pas avec le schéma 1 qui est meilleur que ce second schéma.

- Le processus de transformation d'une relation posant des problèmes lors des mises à jour en relations n'ayant pas ces problèmes, est appelé processus de normalisation.

Normalisation des relations

- Ensemble de règles introduites dans le modèle dès son origine ayant pour but de garantir à la base de données toute sa cohérence lors de manipulation comme l'insertion, la mise à jour, la suppression.
- La normalisation évite les redondances de données.
- Six formes normales (FN) (les trois premières sont les plus importantes) numérotées de 1 à 5 avec une variante pour la troisième.
- Processus important dans la phase de conception. La normalisation permet de définir des relations de telle sortes qu'elles soient l'image d'objets du monde réel en permettant une affectation correcte des attributs dans différentes tables. C'est un outil puissant pour la définition sémantique des relations.
- Normaliser une relation c'est la représenter sous une forme canonique respectant certains critères assurant la définition sémantique de la structure de la BD et l'intégrité des données. Elles a comme conséquence la décomposition de la relation non normalisée en un certain nombre d'autres relations.
- Les FN sont dépendantes ainsi une relation ne peut être en 2 FN que si elle est en 1 FN. A contrario une relation en 5 FN l'est en 4, 3, 2, 1 FN.

Dépendances fonctionnelles

- Dans de nombreuses bases de données, le contrôle de la redondance et la préservation de la cohérence des données sont les plus importants auxquels est confronté le concepteur et l'administrateur.
- La redondance survient quand une information est stockée dans plusieurs endroits. Si ce contenu est modifié, il faut le modifier au niveau de chacune des copies. Si certaines, mais pas toutes les copies, sont modifiées les données sont incohérentes.
- Pour éviter ses écueils, cela passe par l'étude des dépendances fonctionnelles.

Dépendances fonctionnelles

- Définition : Une **dépendance fonctionnelle**, notée DF, indique que la valeur d'un ou plusieurs attributs est associée à *au plus une valeur* d'un ou plusieurs autres attributs.

Dépendances fonctionnelles (DF)

- X et Y deux (ou plusieurs) attributs de la B.D.
- X → Y : Y dépend fonctionnellement de X
- La connaissance de la valeur de X entraîne la connaissance de la valeur de Y.
- X détermine Y.
- Pour une valeur de X, il existe une et une seule valeur de Y.
- Si un attribut (ou un groupe d'attributs) détermine par DF tous les autres attributs de la même relation, c'est une clé de la relation.
- Exemple :
  - Agence → Ville
  - Prêt → Montant
  - Id_article → désignation
  - (Numcom, NumLigne) → Idarticle

Dépendances fonctionnelles

- Les contraintes se classent en deux groupes
  - Les contraintes sémantiques : dépendent de la signification ou de la compréhension des attributs d'une relation
    - Dans une relation Personnel(Nom, Age, Salaire) aucun âge ou salaire ne peut être négatif
  - Les contraintes d'accord ou de concordance ne dépendent pas des valeurs particulières d'un attribut d'un tulpe mais du fait que les tulpes qui acceptent certains attributs acceptent ou non les valeurs de certains de leurs autres attributs
    - Dans une relation Personnel (employé, âge, salaire, service, chef de service) si un employé ne travaille que dans un service et que chaque service n'a qu'un chef de service alors deux tulpes ayant la même valeur dans la colonne chef de service on doit avoir la même valeur dans service. On a une dépendance fonctionnelles
- Les dépendances fonctionnelles sont les plus importante contraintes de concordance ou d'agrément.

Dépendances fonctionnelles

- Définition d'une dépendance fonctionnelle élémentaire
  - une DF, X → B, est une **dépendance fonctionnelle élémentaire** si B est un attribut unique, et si X est un ensemble minimum d'attributs (ou un attribut unique).
- Exemples : Dans la relation Produit, les DF :
  - NP → (couleur, poids) et (NP, NomP) → Poids

  ne sont pas élémentaires.
- Mais les DF :
  - NP → Couleur, NP → Poids, NP → NomP
  - NomP → Couleur, NomP → Poids, NomP → NP

  sont élémentaires.
- La DF :
  - (NP, NF, date) → Qté

  de la relation Livraison est élémentaire.

Dépendances fonctionnelles
Propriétés

- Soient W, X, Y et Z des ensembles d'attributs non vides d'une relation R. Voici quelques propriétés remarquables:
- Réflexivité
  - W est une partie de X alors (X → W)
- Augmentation
  - (W → X) ⟹ (W, Y → X, Y)
- Transitivité
  - (W → X et X → Y) ⟹ (W → Y)
- Union
  - (W → X et W → Y) ⟹ (W → X, Y)
- Pseudo-transitivité
  - (W → X et X, Y → Z) ⟹ (W, Y → Z)
- Décomposition
  - (W → X et Y une partie de X) ⟹ (W → Y)

Première forme normale

- Une relation est en première forme normale si :
  - Elle possède une clé
  - Tous ses attributs sont atomiques : c'est à dire n'ayant à un instant donné qu'une seule valeur ou ne regroupant pas un ensemble de valeurs.
  - Ses éléments sont indivisibles (une seule valeur).
- Si les tables relationnelles résultant de la modélisation ne sont pas déjà en 1FN, il serait approprié de retourner à l'étape de modélisation.
- Une modélisation de qualité minimale devrait toujours être en 1FN.

Un schéma R est en 1NF Si et seulement si les domaines de tous ses attributs sont atomiques

Première forme normale

Exemple :
Clientèle = (IdClient, nom, adresse, tel)

IdClient	Nom	Adresse		Tel
100	Triki	2, rue zerhoune	rabat	776532
101	Alami	5, rue oujda	rabat	756434
102	Badidi	70, rue sebou	casa	346754

Adresse comporte 2 valeurs : adresse et ville, d'où :
Clientèle = (IdClient, nom, adresse, ville, tel) est en 1NF

IdClient	Nom	Adresse	Ville	Tel
100	Triki	2, rue zerhoune	Rabat	776532
101	Alami	5, rue oujda	Rabat	756434
102	Badidi	70, rue sebou	Casa	346754

Deuxième forme normale

- Une relation est en deuxième FN si :
  - Elle est en 1FN
  - Toutes les DF sont élémentaires par rapport à la clé : tout attribut hors clé ne dépend pas d'une partie de la clé

Un schéma R est en 2FN Si et seulement si
Tout attribut de R, n'appartenant pas à la clé primaire , est en dépendance fonctionnelle totale de la clé primaire

Deuxième forme normale

- Exemple
  - Patient (N°patient, Date consultation, Nom)
  - Nom dépend d'une partie de la clé: *N°patient→Nom*
  - Le 2NF permet d'éliminer certaines redondances
    - Patient (N°patient,Nom)
    - Consultation (N°patient*,Date consultation, ordonnance)
- Mais il peut rester des redondances …

Troisième forme normale

- Une relation est en troisième forme normale si :
  - Elle est en 2 FN
  - Tout attribut hors clé est en DF directe par rapport à la clé (pas de transitivité)

Un schéma R est en 3NF ssi

- *R est en 2NF,*
- *Aucun attribut ne dépend transitivement de la clé primaire, (*tout attribut de R, n'appartenant pas à la clé, ne dépend que de la clé*),*

Troisième forme normale

- La 3NF permet d'éliminer des redondances, dues à des dépendances transitives entre attributs mais elle ne suffit pas parfois à éliminer toutes les redondances :
  - Codepostal (Code, Ville, Rue)
  - Les DF sont Code→ Ville et Ville,Rue → Code
  - Cette relation est en 3NF puisque aucun attribut non clé ne dépend d'une partie de la clé ou d'un attribut non clé mais il y a des redondances :

Code	Ville	Rue
50000	Meknes	Marjane
50000	Meknes	Beni M'hammed

Forme de Boyce-Codd (BCNF)

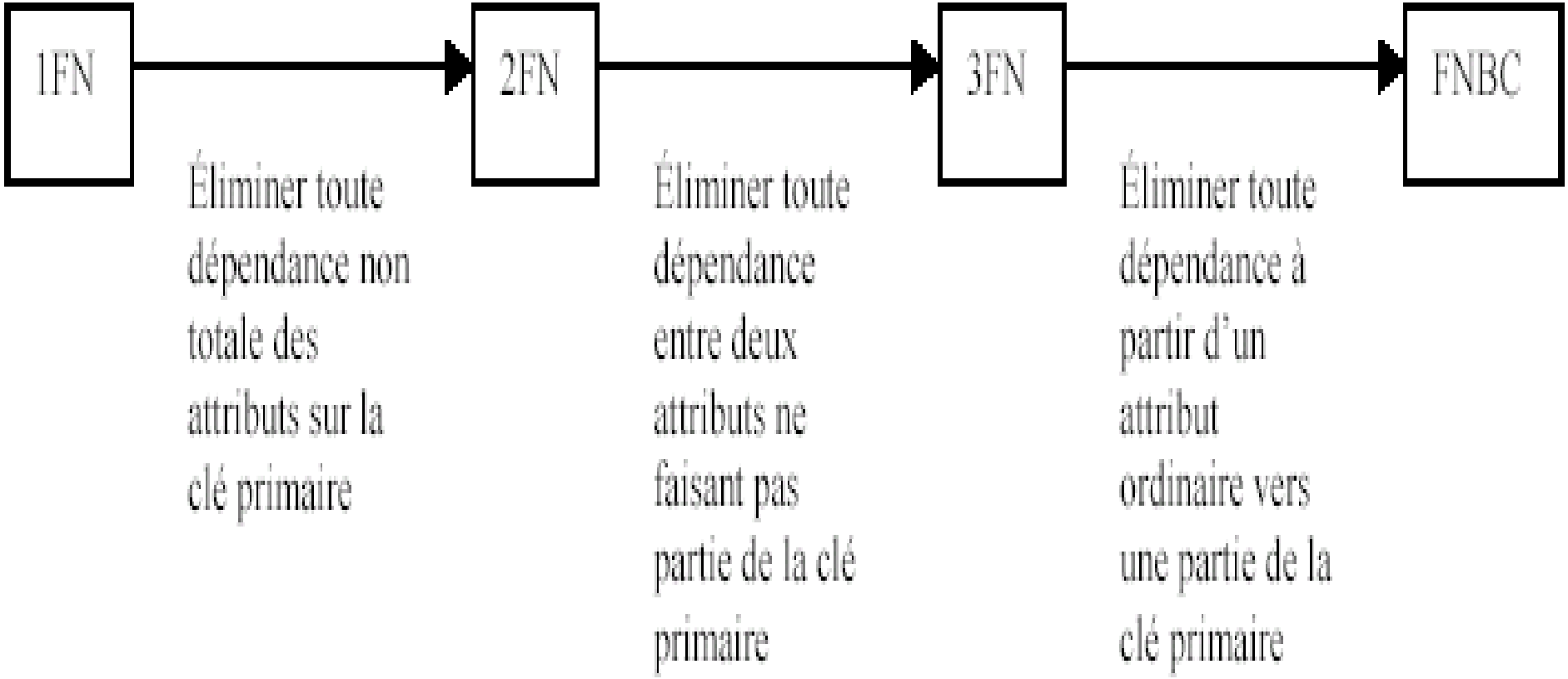
- Une relation est en BCNF si :
  - Elle est en 3 FN
  - Tout attribut non clé de la relation n'est pas source de DF vers une partie de la clé.
- Ou
  - Les seules DF élémentaires qu'elle comporte sont celle où une clé détermine un attribut.
- Dans l'exemple précédant :

  CodeVille (Code*, Ville)

  CodeRue (Code, Rue)

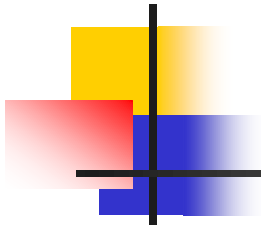
  Sont en BCNF, mais perte de la DF Ville,Rue ->Code
- Toute relation a une décomposition en BCNF sans perte d'information, par contre, une décomposition en BCNF ne préserve pas généralement les DF.

Forme de Boyce-Codd (BCNF)



Méthode de normalisation

- Il est souhaitable qu'un schéma relationnel ne comporte que des relations en 3NF ou BCNF.
- Des algorithmes de constructions permettent d'obtenir de tels schémas. Ils sont de deux catégories :
  - La méthode de décomposition :
    - Elle se base sur la décomposition de relations en utilisant les DF entre les données. Cette méthode conduit à des relations en 3NF ou BCNF. Il y a 2 problèmes :
      - Identification des DF et leurs exhaustivité
      - Le résultat dépend de l'ordre d'application des décompositions et peut ne pas préserver les DF
  - La méthode synthétique
    - Elle se base sur la représentation des DF en terme de graphes (graphe et leur couverture)

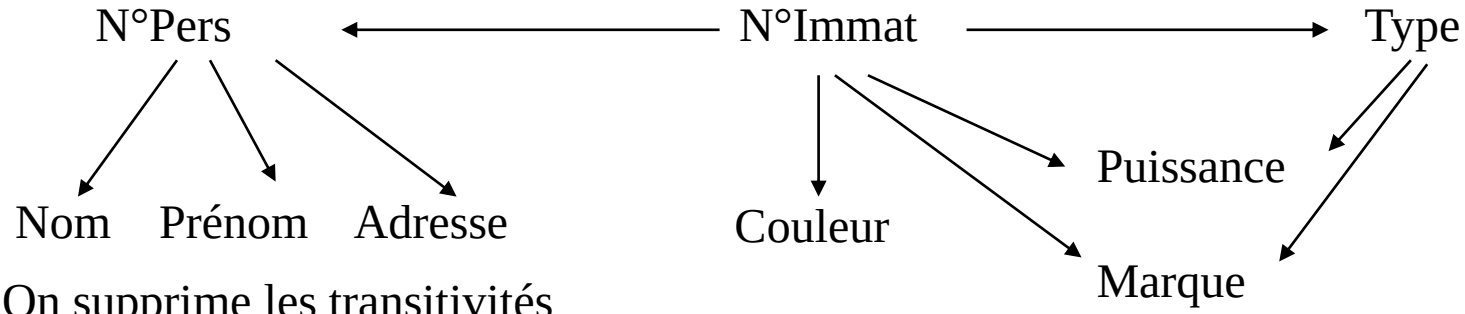


Méthode synthétique

- Point de départ
  - L'ensemble de tous les attributs
  - L'ensemble des DF entre attributs qui sont représentées dans un graphe avec comme nœud un attribut et comme arc une DF
- Ce qu'il faut faire
  - Trouver la couverture minimale du graphe c'est-à-dire éliminer les circuits ainsi que les DF non élémentaires et non directes
- Résultat
  - Une collection de relation en 3NF. Chaque schéma est obtenu en prenant comme :
    - Clé une source de DF
    - Attributs, les buts des DF correspondant

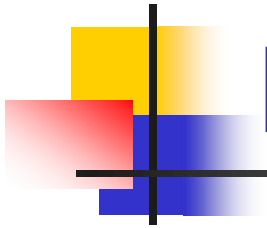
Exemple

- Service d'immatriculation de voitures dans une préfecture
  - Soient les DF suivantes :
    - N°Immat -> Couleur, Type, Puissance, Marque
    - N°Pers -> Nom, Prénom, Adresse
    - N°Immat -> N°Pers et Type -> Marque, Puissance
  - On crée le graphe :



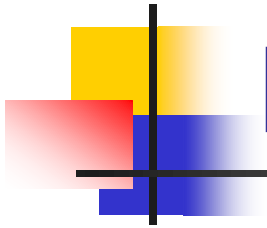
On supprime les transitivités
On obtient :

Personne (N°Pers, Nom, Prénom, Adresse)
Voiture (N°Immat, Couleur, Type*, N°Pers*)
Types (Type, Puissance, Marque)



Exercice

- Soit la relation R(A,B,C,D,E,F,G,H) avec l'ensemble F des dépendances fonctionnelles suivantes:
  - F={ AB→C, B→D, CD→E, CE→GH, G→A}
- Démontrer les affirmations suivantes :
  - AB→EH et BG→GH
- Dessiner le graphe de dépendance
- Donner toutes les clefs condidates de la relation R
- Donner la forme normale de la relation R
- Rendre la relation R en 3FN



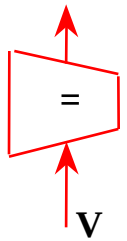
L'algèbre relationnelle

- Elle se compose d'un ensemble d'opérateurs opérant sur des relations et produisant de nouvelles relations.
- Permet de construire de nouvelles informations à partir des relations de départ et d'une composition séquentielle d'opérateurs.
- On peut classifier les opérateurs relationnels en trois catégories :
  - les opérateurs unaires : affectation, sélection et projection
  - les opérateurs binaires travaillant sur des relations de même schéma : union, intersection, différence
  - les opérateurs binaires travaillant sur des relations de schémas différents : jointure, produit cartésien, division

Opérateurs unaires -Selection

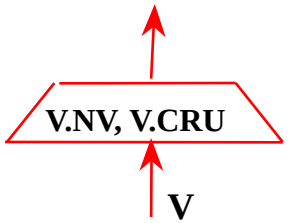
- **sélection** : SELECTION condition-de-sélection (R)

la sélection prend en entrée une relation R définie sur un schéma SR et produit en sortie une nouvelle relation de même schéma SR ayant comme nuplets ceux de R satisfaisant à l'expression de sélection. Une expression de sélection est une condition booléenne construite à partir des connecteurs logiques et, ou, non et de conditions simples



Opérateurs unaires - Projection

- **projection** : PROJECTION A1, ..., An (R)
- la projection prend en entrée une relation R définie sur un schéma SR et produit en sortie une nouvelle relation de schéma A1, ..., An (schéma inclus dans SR) ayant comme nuplets ceux de R restreints au sous-schéma A1, ..., An. Il faut noter que la cardinalité de la nouvelle relation est inférieure ou égale à celle de R, puisque des doublons ont pu être produits par la projection et sont donc supprimés (une relation est toujours un ensemble).



Exemple

- Quels sont les invités du repas du 010597.
- en algèbre :

  R1 <- SELECTION date=010597 (REPAS)

  PROJECTION invité (R1)
- ou bien sous forme fonctionnelle :

  PROJECTION invité (SELECTION date=010597 (REPAS))
- en SQL :

  SELECT distinct invité FROM REPAS

  WHERE date=010597

- Quels sont les plats qui ont été servis à Alice ?
- en algèbre :

  *PROJECTION plat (SELECTION invité=Alice(REPAS) * MENU)*
- en SQL :

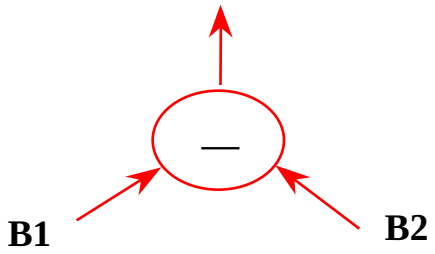
  SELECT distinct plat

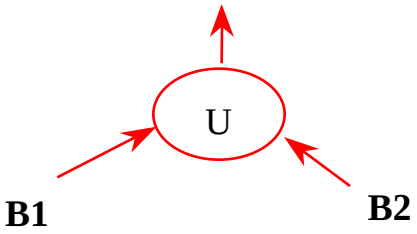
  FROM REPAS R, MENU M

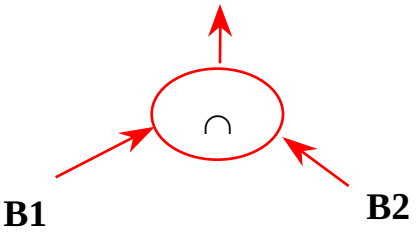
  WHERE R.date=M.date AND invité='Alice'

Opérateurs binaires de même schéma

- **Union :** R ∪ S crée une relation temporaire de même schéma et de population égale à l'ensemble des tuples de R et de ceux de S (avec élimination des doubles éventuellement créés).
- **Différence :** R **-** S crée une relation temporaire de même schéma et de population égale à l'ensemble des tuples de R moins ceux de S, c'est à dire : les tuples qui se trouvent dans R mais pas dans S.
- **Intersection** : R ∩ S crée une relation temporaire de même schéma et de population égale à l'ensemble des tuples de R qui ont un tuple de même valeur dans S



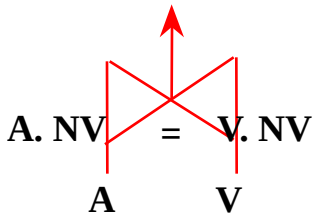


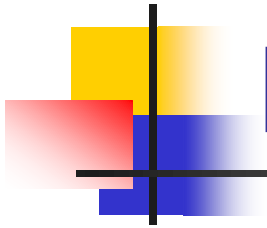


Opérateurs binaires de schémas différents- Jointure

- Définition : étant donné deux relations R(X, Y) et S(Y, Z), où X, Y, Z symbolisent soit un attribut, soit un ensemble d'attributs, et où Y n'est pas vide, la jointure (naturelle) de R et S, notée : R * S

- crée une nouvelle relation temporaire, de schéma (X, Y, Z). La population de R * S est l'ensemble des tuples <x, y, z> créés par composition d'un tuple <x, y> de R et d'un tuple <y, z> de S, tels que les deux tuples ont la même valeur pour Y.





Exemple

- Quels sont les plats qui ont été servis à Alice ?
- en algèbre :

  *PROJECTION plat (SELECTION invité=Alice(REPAS) * MENU)*
- en SQL :

  SELECT distinct plat

  FROM REPAS R, MENU M

  WHERE R.date=M.date AND invité='Alice'

Opérateurs binaires de schémas différents- Division

- Définition : Soient deux relations R (A1, …, An) et V (A1, …, Af) (f < n) telles que tous les attributs de V sont aussi attributs de R, alors la division de R par V, notée : R / V
- crée une nouvelle relation temporaire de schéma (Af+1, Af+2, …, An), et de population égale aux tuples de R, tronqués à [Af+1, Af+2, …, An], et qui existent dans R concaténés à tous les tuples de V,

Exemple

R (A, B, C)

A	B	C
1	1	1
1	2	0
1	2	1
1	3	0
2	1	1
2	3	3
3	1	1
3	2	0
3	2	1

V (B , C)

B	C
1	1
2	0

R / V (A)

A
1
3

V' (B , C)

B	C
1	1

R / V' (A)

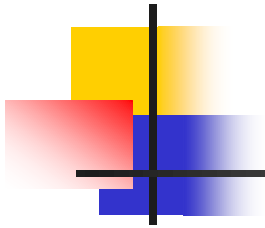
A
1
2
3

V'' (B , C)

B	C
3	5

R / V'' (A)

/



Exemple

- Quels sont les invités qui sont venus à tous les repas ?

en algèbre :

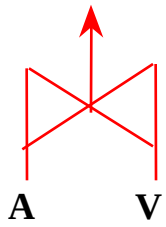
REPAS / PROJECTION date (REPAS)

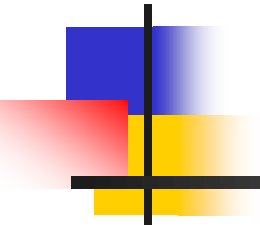
- en SQL :

```
SELECT invité FROM REPAS
GROUP BY invité
HAVING count(*) =
(SELECT count(distinct date)
FROM REPAS)
```

Opérateurs binaires de schémas différents- Produit cartésien

- Définition : Soient deux relations, R (A1, A2, …, An) et T (B1, B2, …, Bp), n'ayant pas d'attribut de même nom, alors le produit de R par T, noté R x T, crée une relation temporaire de schéma (A1, A2, …, An, B1, B2, …, Bp) et de population toutes les concaténations possibles de tuples de R et de T





LE LANGAGE SQL

Langage de manipulation de données (LMD)

Syntaxe complète de l'interrogation

SELECT {[**ALL** | **DISTINCT**] <colonne et/ou calculs> [AS <alias>]} | *

FROM <table1 [var1], table2 [var2] …>

[**WHERE** < condition de sélection >]

[**GROUP BY** <colonne1 , colonne2…>

[**HAVING** <critères_regroupement>]]

[{**UNION** | **INTERSECT** | **MINUS**} (**SELECT** …)]

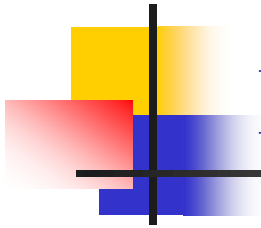
[ORDER BY <colonne1 [ASC | DESC]**, …>]**

Ligne SELECT

SELECT {[**ALL | DISTINCT**] <colonne et/ou calculs> [AS <alias>]} | *

- **ALL** : sans élimination des doublons (mode par défaut)
- **DISTINCT** : avec élimination des doublons

- **colonne :**
  - soit nom_att (si pas d'ambiguïté sur ce nom),
  - soit nom_table.nom_att,
  - soit * (toutes les colonnes),
  - soit nom_tab.*



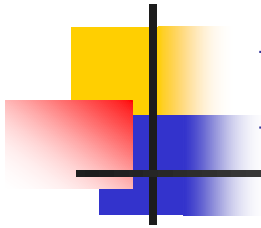
Ligne SELECT

SELECT {[**ALL** | **DISTINCT**] <colonne et/ou calculs> [**AS** <alias>]} | *

calcul :

- soit expression sur des colonnes avec des opérateurs : +, -, *, / éventuellement avec (), ou || (concaténation de chaînes de caractères)
- soit des fonctions simples sur une valeur ou agrégats sur un ensemble de valeurs (ex: SUM (somme), AVG (moyenne) des valeurs d'une colonne, COUNT (compte les n-uplets))
- soit la combinaison des expressions, fonctions simples et agrégats (ex: MIN, MAX d'une expression sur des colonnes)

alias : si mentionné, alias est le nom de la colonne dans le résultat de la requête.



Ligne FROM

FROM <table1 [var1], table2 [var2] …>

table1 :

- soit un nom de table,
- soit un nom de table préfixé du compte du propriétaire

var1 : variable (alias) attribuée à table1 durant la requête

Ligne WHERE (Optionnelle)

[**WHERE** <condition de sélection>]

condition : expression logique de prédicats

- colonne *comparateur* { valeur | colonne | sous-requête}
  où le comparateur est l'un des suivants :
  =, !=, >, <, >=, <=, LIKE, {ALL| ANY}
- colonne {IN | NOT IN} { (valeur1, valeur2,…) | sous-requête}
- colonne BETWEEN valeur1 AND valeur2
- {EXISTS | NOT EXISTS} sous_requête
- colonne IS NULL
- condition {AND|OR} condition,
- NOT condition

Ligne GROUP BY (Optionnelle)

[**GROUP BY** <colonne1 , colonne2…>
[**HAVING** <critères_regroupement>]]

- colonnes : attributs de regroupement
- objectif : partitionner horizontalement la relation selon les valeurs de certaines colonnes, appliquer ensuite une fonction agrégat (sur d'autres colonnes) aux partitions
- **Ligne HAVING** (Optionnelle après regroupement)

critère_regroupement :

fonction_agrégat([DISTINCT] colonne) *comparateur* {valeur |sous-requête}

Opérations ensemblistes

- **UNION :** effectue l'union des n-uplets de deux relations
- **INTERSECT :** effectue l'intersection des n-uplets de deux relations
- **MINUS :** effectue la différence ensembliste entre deux relations

Remarque : les attributs des lignes SELECT doivent être les mêmes pour pouvoir appliquer ces opérations entre deux requêtes

Ligne ORDER BY (optionnelle)

[ORDER BY <colonne1 **[**ASC | DESC**], …>]**

ordonne le résultat de la requête dans l'ordre croissant (ASC) ou décroissant (DESC) des valeurs de la colonne1…

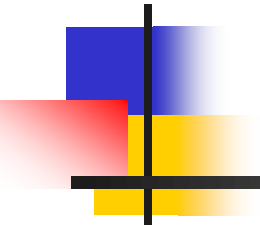
Ligne GROUP BY (Optionnelle)

[**GROUP BY** <colonne1 , colonne2…>
[**HAVING** <critères_regroupement>]]

- colonnes : attributs de regroupement
- objectif : partitioner horizontalement la relation selon les valeurs de certaines colonnes, appliquer ensuite une fonction agrégat (sur d'autres colonnes) aux partitions
- **Ligne HAVING** (Optionnelle après regroupement)

critère_regroupement :

fonction_agrégat([DISTINCT] colonne) *comparateur* {valeur |sous-requête}



LE LANGAGE SQL

Langage de Définition de données (LDD)

Création de table

I. Requêtes Actions

⇩ Sont des requêtes qui permettent de **créer** des tables, d'**ajouter,** de **supprimer** des enregistrements d'une table, d'ajouter une colonne…

⇩ L'instruction **CREATE TABLE** permet de créer une nouvelle table

Syntaxe

1. Création d'une table avec un seul champ comme clé primaire

CREATE TABLE Nom_table **(champ1** type **CONSTRAINT** nom_contrainte PRIMARY KEY**, champ2** type **[NOT NULL], …, champN** type **[NOT NULL]);**

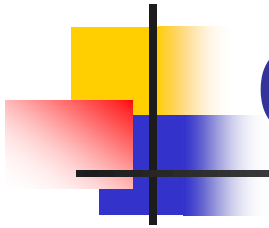
Création de table

2. Création d'une table avec plusieurs champs comme clé primaire

CREATE TABLE Nom_table **(champ1** type **[NOT NULL], champ2** type **[NOT NULL], …, champN** type**,** **CONSTRAINT** nom_contrainte PRIMARY KEY **(**champ1, champ2,…**)));**

/* Créez une requête SQL permettant de créer la table EMPLOYES (Nemployé, Nom, Prénom, Fonction, Adresse, Codeville) tels que le 1er champ est de type entier les autres de type texte ayant respectivement une taille de: 25, 20, 15 et 50. Tandis que le champ Codeville est de type Entier long */

CREATE TABLE EMPLOYES **(** Nemployé INTEGER **CONSTRAINT** nom_index **PRIMARY KEY,** Nom TEXT(25), Prénom TEXT(20), Fonction TEXT(15), Adresse TEXT(50), Codeville LONG) **) ;**



Création de table

⇩ Les conventions relatives aux noms des tables et des champs varient quelque peu d'un SGBD à l'autre. En ce qui concerne plus particulièrement les champs:

- Le nombre de caractères ne doit pas être trop grand (64 dans Access, 18 à 30 dans d'autres SGBD) ;
- Seuls les lettres, les chiffres et le caractère de soulignement sont autorisés. Access admet les caractères accentués. Il admet aussi l'espace, mais le nom du champ doit alors être écrit entre crochets ;
- Certains SGBD requièrent que le nom d'un champ commence par une lettre, mais ce n'est pas le cas d'Access ;

Création de table

- Les termes faisant partie du vocabulaire du langage SQL sont interdits ("date" par exemple). Ce sont les mots réservés.

⇩ Voici un échantillon représentatif des différentes façons d'exprimer un type de données lors de la création d'une table en SQL dans Access :

- Booléen (Oui / Non) : **BIT** ;
- Nombre entier : **SHORT** (entier), **SMALLINT** (entier), **LONG** (entier long), **INTEGER** (entier long), **BYTE** (octet) ;
- Nombre réel : **SINGLE** (réel simple), **DOUBLE** (réel double), **NUMERIC** (réel double) ;

Création de table

- Monétaire : **CURRENCY**, **MONEY** ;
- Date/Heure : **DATE**, **TIME**, **DATETIME** ;
- Texte : **VARCHAR** (255 caractères), **CHAR(n)** ou **TEXT(n)** (n caractères), **LONGTEXT** (mémo, 65 535 caractères =32K max.) ;
- Fichier binaire : **LONGBINARY** (Objet OLE) ;
- Compteur : **COUNTER** (NuméroAuto).

Remarque:

On notera qu'il n'est pas possible de créer un champ de type **hypertexte** via une commande SQL dans Access. Même remarque en ce qui concerne les **listes de choix.**

Création de table

/* Créez une requête SQL permettant de créer la table DETAILS (Ncommande, Réf, Pu,Quantité, Remise) tels que les champs sont de type respectivement: entier long, texte de taille 30, monétaire, entier et réel simple et tel que aussi les champs Pu & Quantité sont non nuls*/

```
CREATE TABLE DETAILS ( Ncommande Integer, Réf  Char(30),
   Pu  Currency NOT NULL, Quantité Smallint NOT NULL, Remise Single,
  CONSTRAINT nom_index PRIMARY KEY ( Ncommande, Réf ) ) ;
```

Création de table

/* Créez une requête SQL permettant de créer la table VILLES (Codeville, Ville) tels que le 1er champ est de type NumérоAuto et le deuxième est de type texte ayant une taille de 20*/

```
CREATE TABLE VILLES ( Codeville  Counter CONSTRAINT nom_index
    PRIMARY KEY, Ville Text(20) ) ;
```

ALTER Table

- L'instruction **ALTER TABLE** permet d'ajouter ou de supprimer un **seul** champ à une table. Elle permet aussi la création et la suppression des liens entre les tables d'une base de données.

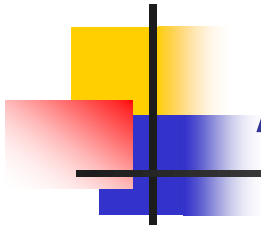
ALTER Table

/* Modifiez la table EMPLOYES en déclarant le champ "CodeVille" comme clé étrangère, puis créez un lien nommé lien_ville sur le champ CodeVille, en précisant que le côté 1 du lien est le champ Code_Ville de la table VILLES .*/

ALTER TABLE EMPLOYES **ADD CONSTRAINT** Lien_ville
FOREIGN KEY (Code_Ville) **REFERENCES** VILLES (Code_Ville);

/* Supprimer le lien nommé lien_ville existant entre la table EMPLOYES et la table VILLES selon le champ Codeville.*/

ALTER TABLE EMPLOYES **DROP CONSTRAINT** Lien_ville;



ALTER Table

/* Créez une requête SQL permettant d'ajouter le champ Codecli à la table DETAILS. Ce champ est de type texte, de taille 20 caractères et il est non null

```
ALTER TABLE DETAILS ADD COLUMN Codecli TEXT(20) NOT NULL;
```

/* Créez une requête SQL permettant de supprimer le champ Codecli de la table DETAILS.

```
ALTER TABLE DETAILS DROP COLUMN Codecli ;
```

DROP TABLE

⇩ L'instruction **DROP TABLE** permet de supprimer une table d'une base de données

Syntaxe

DROP TABLE Nom_table **;**

/* Créez une requête SQL permettant de supprimer la table DETAILS définitivement de la base de données

DROP TABLE DETAILS **;**

Soit la table **ACTEURS** créée sous ***ACCESS***:

ACTEURS (**N_act**, Nom, Prénom, Nationalité, Salaire, Age, Films)

DELETE

⇩ L'instruction **DELETE** permet d'**effacer** des **enregistrements** d'une table

/* Effacez tous les enregistrements de la table ACTEURS*/

DELETE * **FROM** ACTEURS **;**

(Ou bien)

DELETE N_act **FROM** ACTEURS **;**

/* Effacez tous les acteurs de nationalité marocaine*/

DELETE * FROM ACTEURS **WHERE** Nationalité= "marocaine"**;**

UPDATE

⇩ L'instruction **UPDATE** permet la mise à jour d'une table

⇩ **Syntaxe :** **UPDATE** table **SET** nouvellesvaleurs WHERE critères **;**

/* Créez une requête permettant de modifier l'adresse de l'employé numéro 10 tout en sachant que la nouvelle adresse sera "10 Avenue Mohamed VI, Tanger"*/

```
UPDATE EMPLOYES SET Adresse = "10 Avenue Mohamed VI, Tanger"
WHERE Nemployé = 10 ;
```

/* Créez une requête permettant d'augmenter de 3% le salaire de tous les acteurs */

```
UPDATE ACTEURS SET Salaire = Salaire * 1.03 ;
```

INSERT INTO

L'instruction **INSERT INTO** permet d'ajouter un ou plusieurs enregistrements à une table

Syntaxe

1. **Requête ajout d'un seul enregistrement**

```
INSERT INTO  Nom_table [(champ1, champ2, …)]
VALUES (valeur1, valeur2, …) ;
```

2. **Requête ajout de plusieurs enregistrements**

```
INSERT INTO  Nom_table 1 [IN externaldatabase] (Champ1, Champ2, …)
   SELECT (Champ1, Champ2, …)
   FROM Nom_table 2 ;
```

INSERT INTO

/* Créez une requête permettant d'ajouter l'enregistrement suivant dans la table EMPLOYES: (100, BEN AZOUZ, Aziz, Ingénieur, 90050) */

INSERT INTO EMPLOYES (Nemployé, Nom, Prénom, Fonction, Codeville)

VALUES (100, "BEN AZOUZ", "Aziz", "Ingénieur", 90050) ;

/* Soit la table **NOUVEAUX_EMPLOYES** (**Nemployé**, Nom, Prénom, Fonction, Adr, Codeville) Supposons que cette table contient des enregistrements**. Question: Ajoutez tous les enregistrements de la table ci-dessus dans la table EMPLOYES*/**

INSERT INTO EMPLOYES

SELECT * FROM **NOUVEAUX_EMPLOYES ;**