IUP GMI D'AVIGNON

www.centraliup.fr.st

Systèmes de Gestion de Base de Données

Thierry Spriet

Version rééditée au format PDF et remise en page

Table des matières

1. GENERALITES SUR LES S.G.B.D

1.1. DEFINITION

BASE DE DONNEES : Collection de données dont la structure reflète les relations qui existent entre ces données. Cette base de données est gérée par un Système de Gestion de Base de Données (SGBD).

S.G.B.D : C'est un ensemble de procédures permettant :

- La description des données et des relations les concernant.
- L'interrogation de la base de données (ex: SNCF utilise le système SOCRATE).
- La mise à jour des données (éventuelle redondance de données).
- Le partage des données.
- La protection des données de la base.

1.2. HISTORIQUE

- **Années 50-60** : A cette époque, des objets informatiques ont été structurés : des fichiers, avec des modes d'accés à ces fichiers (séquentiel, séquentiel indexé, direct ...).

- **Années 62-63** : Apparition du concept de base de données.

- **Années 65-70** : SGBD première génération (i.e : modèle hiérarchique [arbres] et réseau [graphes]).

  - IMS d'IBM (hiérarchique).
  - IDS de General Electric (réseau), a servi de modèle pour CODASYL.

- **A partir des années 70** : SGBD seconde génération fondée sur le modèle relationnel. Avantages : plus de spécification des moyens d'accés aux données.

- **Années 80** :

  - MRDS (CII HB).
  - QBE (Query By Example).
  - SQL / IDS (IBM).
  - INGRES.
  - ORACLE (choisi pour ce cours).

- **Futur** :
  - Les données seront plus variées (textes, sons, images...).
  - Bases de connaissances. Systèmes experts.
  - Bases de données déductives.
  - Génie logiciel et SGBD.
  - Accés intelligent et naturels (langage naturel par exemple)
  - Communication multimédia.

1.3. OBJECTIFS D'UN SGBD IDEAL

• Indépendance physique (données / programmes) : modifier la structure interne des données, sans toucher le(s) programme(s).

• Indépendance logique : pouvoir modifier le schéma conceptuel sans modifier les programmes (lors de rajout d'informations : tables).

• Manipulation des données : il faut que les données puissent être manipulées à distance par des gens qui n'ont aucune connaissance de la structure interne de la base de données.

• Efficacité des accés aux données :

- Convivialité : manipuler les données sans connaitre les structures (utilisateurs).
- Rapidité d'accés aux données : temps de réponse.

• Administration centrale des données : organisation générale et administration de la base.

• Non-redondance des données : une même donnée ne devrait apparaître qu'une seule fois dans la base (il existe des cas où cela est impossible).

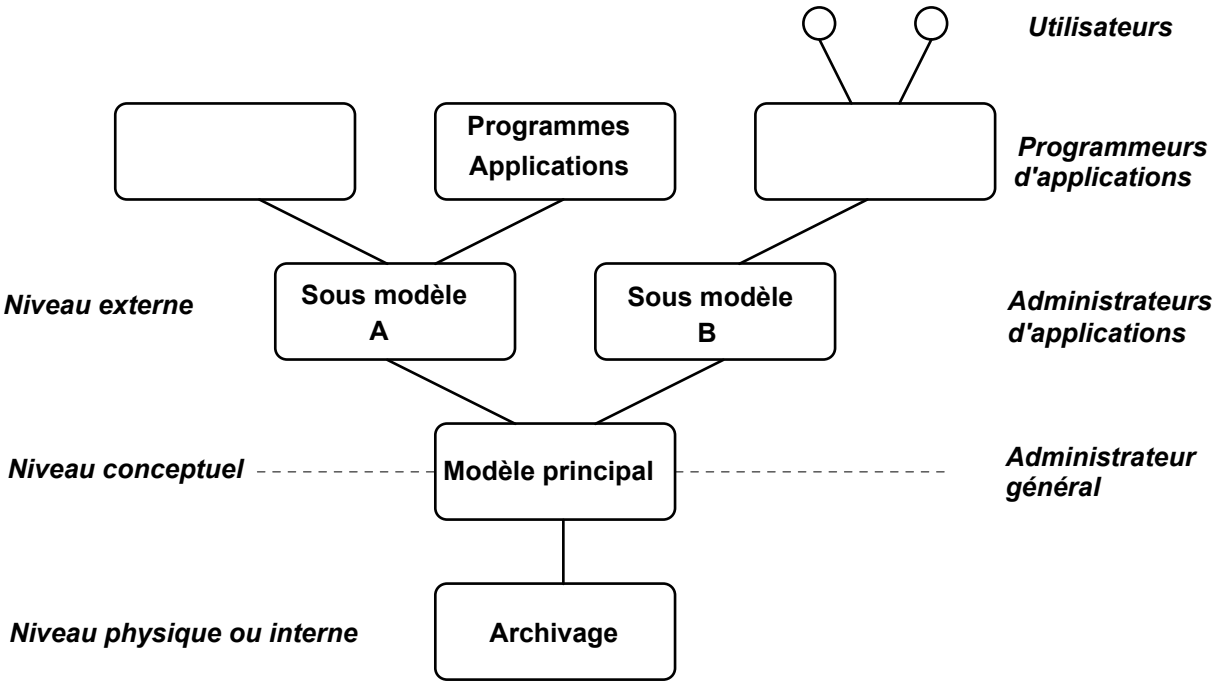
• Intégrité des données : l'administrateur, lorsqu'il définit des données, détermine des contraintes sur l'intégrité des données.

• Partageabilité des données.

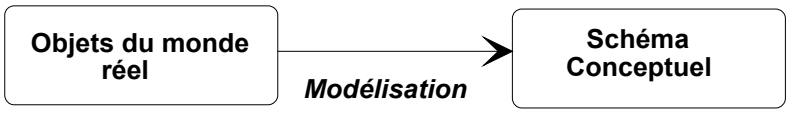
• Sécurité des données :

- Contrôles des droits d'accés.
- Reprises sur pannes.

1.4. ARCHITECTURE D'UN SGBD



1.4.1. NIVEAU CONCEPTUEL



On dispose pour cela d'un Langage de Définition de Données (LDD) qui permet de décrire le schéma conceptuel de notre base.

Notamment :

- Définir et nommer les catégories d'objets.
- Définir et nommer les relations entre objets.
- Exprimer des contraintes sur les données.

1.4.2. NIVEAU INTERNE (ou PHYSIQUE)

Spécifications du stockage physique des données (fichiers, disques, etc.) et des méthodes d'accès (index, chaînages, etc.).

1.4.3. NIVEAU EXTERNE

C'est une vue externe pour chaque groupe d'utilisateurs sur un sous-ensemble de la base.
Le schéma externe est généralement un sous schéma du schéma conceptuel mais il peut contenir des informations supplémentaires (non prévues dans le schéma général, mais nécessaires à l'application spécifique de celui-ci).

1.5. LES UTILISATEURS DE LA BASE

1.5.1. L'ADMINISTRATEUR PRINCIPAL

- Définit le schéma conceptuel.
- Conditionne l'évolution de la base.
- Définit les modalités d'accès et de protection des données.

1.5.2. LES ADMINISTRATEURS D'APPLICATIONS

- Définit le sous modèle adapté à l'application.
- Elabore les schémas externes (tables qui seront accessibles aux utilisateurs).
- Définit des règles de correspondance entre schéma externe et schéma interne.

✓ C'est à partir des travaux des administrateurs que va s'élaborer le **dictionnaire des données.** Il correspond à la mémoire conservée de tous les objets ayant été créés. Ce dictionnaire permet de faire des statistiques d'après coups et ne sera jamais effacé.

1.5.3. LES PROGRAMMEURS D'APPLICATIONS

Ils réalisent des bibliothèques de programmes pour la manipulation et le traitement des données (interrogation, mise à jour, ...). On utilise pour cela un Langage de Manipulation de Données (LMD).

Exemple : langage SQL.

2. MODELES LOGIQUES DE DONNEES

2.1. MODELE CONCEPTUEL

Objets, liens et entités (technique OLE).

2.1.1. OBJETS

Un objet est la plus petite abstraction (données de même type) qui a une signification pour l'utilisateur.

Exemple :

AV#
AVION : nom de l'avion (B737, A320, ...).
AVCAP : capacité de l'avion (200, ...).
AVLOC : ville d'attache de l'avion.
AVREV : date de dernière révision.

PL#
PLNOM : nom de pilote.
PLADR : adresse d'un pilote (atomique, si l'adresse n'était pas restreinte à un nom de ville, on aurait une entité).

VOL# : numéro de vol (un vol a un et un seul numéro).

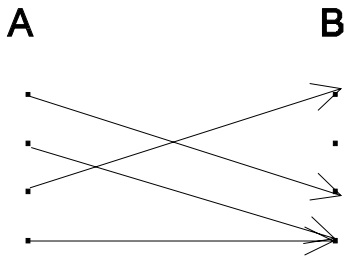
V_D : ville de départ.
V_A : ville d'arrivée.

H_D : heure de départ.
H_A : heure d'arrivée.
.
.
.

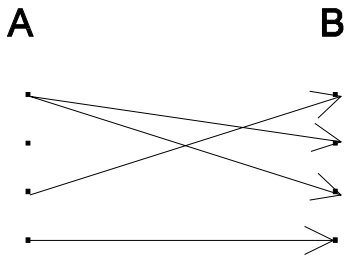
2.1.2. LIENS

Un lien est une association entre objets qui traduit une contrainte de l'entreprise (au sens le plus large).

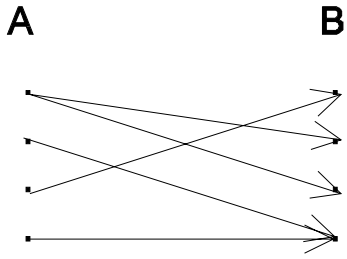
- **Lien de type N:1 (liens fonctionnels)**



- **Lien de type 1:N (liens hiérarchiques)**



- **Lien de type N:M (liens maillés)**



Exemple 1 : liens de type N:1

VOL# ⟶ (V_D, V_A)
IB102⟶ (Toulouse, Madrid)
IB104 ⟶ (Toulouse, Barcelone)
IB106 ⟶ (Toulouse, Barcelone)
IB108 ⟶ (Marseille, Nice)

Exemple 2 : Liens de type 1:N

Date de naissance ⟶ Personne

29/11/92 ⟶ Paul
29/11/92 ⟶ Martine
18/04/72 ⟶ Jacques

Exemple 3 : Liens de type N:M

Enseignement ⟶ Module

Licence math ⟶ Prog
IUP1 ⟶ Prog
DESS DC ⟶ Prog
IUP1 ⟶ Anglais

2.1.3. ENTITES

Une entité est constituée d'objets et de liens fonctionnels. C'est un ensemble d'objets liés fonctionnellement (N:1) qui représente une abstraction de l'entreprise et qui peut être nommée.

Exemple : PILOTE est une entité constituée de manière unique d'un ensemble d'attributs (objets / entités) tels que PLNOM, PLNUM et PLADR.

- **Entités statiques**

Il n'y a que des attributs de type objet.

Exemple : L'entité AVION est composée des objets AVNOM, AVCAP et AVLOC.

- **Entités dynamiques**

Dépendent d'autres entités statiques ou dynamiques.

Exemple : L'entité VOL est composée des objets VOL#, V_D, V_A, H_D et H_A. Son existence est également liée aux entités AVION et PILOTE.

2.1.4. EXEMPLE

Entités : VOL, AVION et PILOTE.

Objets : VOL#, V_D, V_A, H_D et H_A.
AV#, AVNOM, AVCAP et AVLOC.
PL#, PLNOM et PLADR.

Liens sémantiques intra et inter entités :

❶ Chaque VOL, AVION ou PILOTE est déterminée de manière unique par son numéro.

VOL# — N:1 → PL#, AV#, V_D, V_A, H_D, H_A
PL# — N:1 → PLNOM, PLADR
AV# — N:1 → AVNOM, AVCAP, AVLOC

❷ Chaque avion peut être conduit par plusieurs pilotes.

AVION — N:M → PILOTE

❸ Un pilote peut assurer plusieurs vols.

PILOTE — 1:N → VOL

❹ Un avion peut avoir un ou plusieurs vols.

AVION — 1:N → VOL

2.2. MODELE HIERARCHIQUE

Représentation du schéma sous la forme d'une arborescence. On va être obligé de dupliquer des informations car quelquefois, on ne pourra pas accéder à une information sans en connaître les parents.

2.2.1. REPRESENTATION DU SCHEMA

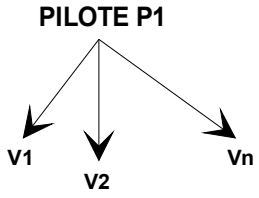
Les liens sont exclusivement du type 1:N.
Le schéma est exprimé au moyen d'un arbre ordonné (arborescence).

Soient les entités : AVION
VOL (résulte de la composition des entités AVION et PILOTE)
PILOTE

On avait : AVION — N:M → PILOTE
PILOTE — 1:N → VOL
AVION — 1:N → VOL

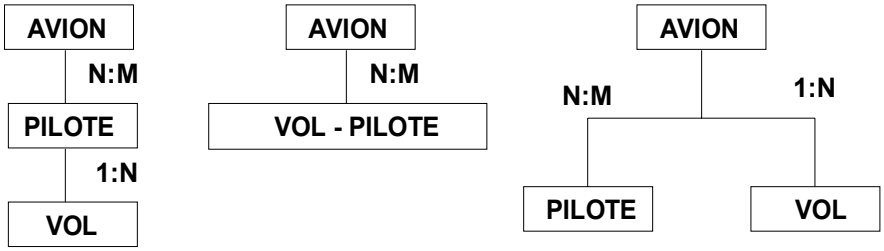
Représentation hiérarchique :

Problème : Avec le modèle hiérarchique, il faut dupliquer AVION pour y accéder (problème de la relation N:M).

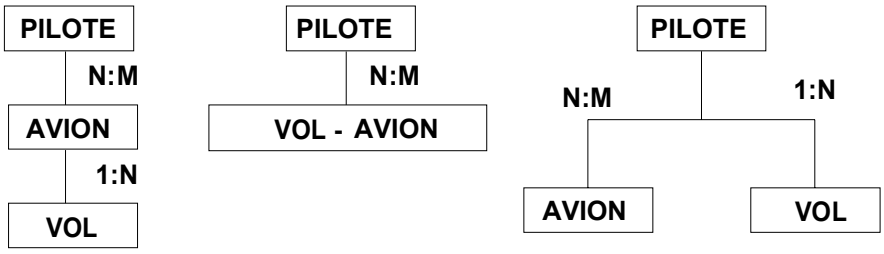


On peut choisir comme point de départ l'entité AVION ou PILOTE.

- Point de départ : AVION



- Point de départ : PILOTE



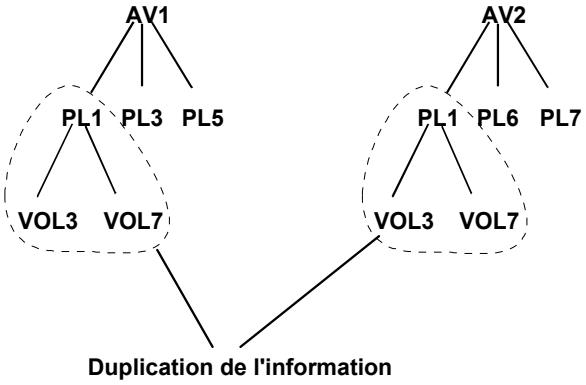
Ces schémas caractérisent la base de données.

Par la suite, on aura :

AV1
AV2
.
.
.
AVn

PL1
PL2
.
.
.
PLn

VOL1
VOL2
.
.
.
VOLn



2.2.2. AVANTAGES DU MODELE HIERARCHIQUE

- Adéquation du modèle avec les entreprises à structure arborescente.

- Simplicité du modèle et de son implémentation (structure d'arbre, les plus communément utilisées).

- Adéquation entre la structure du schéma et les besoins des utilisateurs.

2.2.3. INCONVENIENTS DU MODELE HIERARCHIQUE

- Impossibilité de représenter des liens de type N:M (entraînant des redondances).

- Anomalies dans les opérations de stockage.
  - ❒ La suppression d'un noeud entraîne la disparition de tous les descendants (problème de réécriture d'informations afin de maintenir les informations dans la base).
  - ❒ L'insertion d'une information demande la création d'un segment parent (exemple : Ajouter un vol sans en connaître encore le pilote).
  - ❒ Le remplacement d'une information doit s'effectuer pour toutes ses occurrences dans les arbres distincts.

- Chemin d'accès unique aux données.

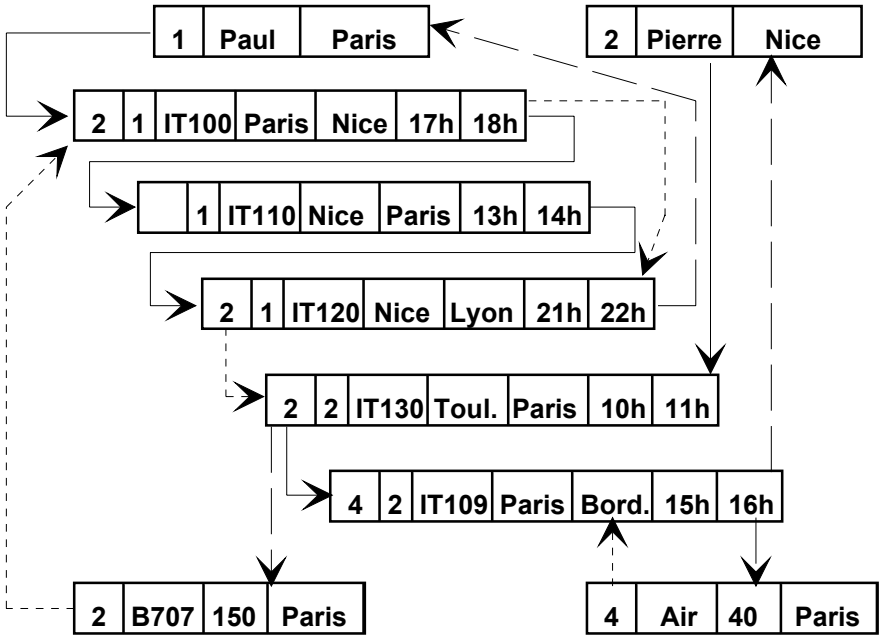
- Indépendance logique très réduite.

2.3. MODELE RESEAU

C'est un simple développement du modèle hiérarchique.

2.3.1. REPRESENTATION DU SCHEMA

Le modèle privilégie les liens de type 1:N. Il permet une représentation symétrique des liens de type N:M. Le schéma est exprimé sous la forme d'un graphe.



2.3.2. AVANTAGES DU MODELE RESEAU

- Permet de représenter les liens N:M.
- Elimination des redondances de données.
- Création d'accès multiples aux données.
- Absence d'anomalies dans les opérations de stockage.

2.3.3. INCONVENIENT DU MODELE RESEAU

- Pas d'indépendance vis à vis des stratégies d'accès (pas d'accès direct).

2.4. MODELE RELATIONNEL

Introduit par CODD chez IBM en 1970.

3. LE MODELE RELATIONNEL

3.1. DEFINITIONS

3.1.1. DOMAINE

Un domaine est un ensemble de valeurs.

Exemples : Domaine des villes : V = {Paris, Nice, Avignon, ... } = D2
Domaine des noms : N = {Jean, Paul, Pierre, ... } = D1

3.1.2. RELATION

Une relation est un sous ensemble du produit cartésien d'une liste de domaines.

$$R \subset D1 \times D2 \times D3 \times ... \times Dn$$

Exemples : ***Attribut*** ↓

HABITE	D1	D2
	Paul	Nice
	Martine	Nice
	Pierre	Paris
Tuple →	Jacques	Avignon

RELIE	D2	D2
	Avignon	Nice

3.1.3. ATTRIBUT

Un attribut est une colonne d'une relation caractérisée par un nom.

La première table ci-dessus définit la relation en ***extension*** (ensemble de tous les cas possibles à un instant donné).
La relation est définie en ***intention*** au moyen d'un schéma qui contient le nom de la relation et la liste des attributs.

Exemple : HABITE (NOM, VILLE)

3.1.4. BASE DE DONNEES RELATIONNELLE

Une base de données relationnelle est un ensemble de schémas de relations.

3.2. CONCEPTION DE SCHEMAS RELATIONNELS

Dans une SGBD relationnelle, les entités et les liens sont représentés par des relations.

Exemple 1 : L'entité AVION est représentée par :
AVION (NUM_AV, NOM_AV, CAP_AV, LOC_AV)

Exemple 2 : Le lien existant entre les entités PILOTE et AVION est VOL
VOL (PILOTE, AVION, V_D, V_A, H_D, H_A)

3.2.1. PROBLEMES DE REPRESENTATION DU REEL

VOL										
IT12	PIL01	Paul	Nice	AV05	B747	500	Nice	Paris	12h	13h
IT14	PIL01	Paul	Nice	AV05	B747	500	Nice	Paris	17h	18h
IT15	PIL02	Yves	Pau	AV12	A300	300	Nice	Paris	7h	8h

3.2.2. APPROCHE PAR DECOMPOSITION

La relation universelle constituée de tous les attributs est décomposée en relations qui n'ont pas les anomalies précédentes.
Cette décomposition nécessite deux opérations : - La projection
- La jointure naturelle

✪ La projection

La projection consiste à supprimer des attributs d'une relation et à éliminer les tuples en double qui peuvent apparaître dans la nouvelle relation.

La projection de R (A1, A2, ... , An) sur les attributs Ai_1, Ai_2, ... Ai_p , avec $i_l \not\subset i_k$, est la relation R' (Ai_1, Ai_2, ... $Ai_{p)}$ et notée Π Ai_1, Ai_2, ... Ai_p (R).

Exemple : Π NUM_AV, NOM_AV, CAP_AV, LOC_AV (VOL)

NUM_AV	NOM_AV	CAP_AV	LOC_AV
AV05	B747	500	Nice
AV12	A300	300	Nice

✪ La jointure naturelle

La jointure naturelle de relations R (A1, A2, ... , An) et S (B1,B2, ... , Bp) est une relation T ayant pour attributs l'union des attributs de R et de S, et pour tuples tous ceux obtenus par concaténation des tuples de R et de S ayant même valeur pour les attributs de même nom.
De plus, la projection sur A1, A2, ... , An (T) est R, et Π B1, B2, ... , Bp (T) = S.

Exemple :

R	
AV01	B747
AV02	B747
AV12	A300
AV13	A300

S		
B747	500	Nice
B747	500	Paris
A300	300	Paris

R⊠S	NUM_AV	NOM_AV	CAP_AV	VILLE
	AV01	B747	500	Nice

AV01	B747	500	Paris
AV02	B747	500	Nice
AV02	B747	500	Paris
AV12	A300	300	Paris
AV13	A300	300	Paris

✪ Propriétés des jointures naturelles

- La jointure naturelle est *associative* :
  (R⊠S) ⊠ T = R⊠ (S⊠T)

- La jointure naturelle est *commutative* :
  R⊠S = S⊠R

3.2.3. DECOMPOSITION

La décomposition d'une relation R(A1, A2, ..., An) est le remplacement de R par une collection de relations R1, R2, ... , Rp obtenues par projection de R et telles que R1⊠R2⊠R3⊠ ... ⊠Rp ont même schéma que R.

Une décomposition est dite sans perte, si pour toute extension de R, on a :

R = R1⊠R2⊠R3⊠ ... ⊠Rp

3.3. DEPENDANCES FONCTIONNELLES

Va servir à caractériser des relations qui peuvent être décomposées sans pertes.

N.B. : On montrera que s'il y a dépendance fonctionnelle, on peut décomposer sans pertes.

3.3.1. DEPENDANCE FONCTIONNELLE

Soit R(A1, A2, ... , An) et X et Y des sous ensembles d'attributs de {A1 ,A2, ... , An}.
On dit que X détermine Y (ou que Y dépend fonctionnellement de X), que l'on note $(X \rightarrow Y)$, si pour toute extension r de R, pour tout tuples t1 et t2 de r, on a :

Π X (t1) = Π X (t2) $\Rightarrow$ Π Y (t1) = Π Y(t2)

Exemples : NUM_PL $\rightarrow$ NOM_PL (le nom du pilote dépend du numéro de pilote)

NUM_PL $\rightarrow$ ADR_PL

(NUM_AV, H_D, H_A) $\rightarrow$ (V_D, V_A)

(NUM_AV, V_D, V_A) $\not\rightarrow$ (H_D, H_A) (car l'avion peut faire un même trajet plusieurs fois).

Remarque : Les dépendances fonctionnelles concernent l'intention des relations. (Elles ne dépendent pas des tables).

3.3.2. PROPRIETES DES DEPENDANCES FONCTIONNELLES

- Réflexivité : $Y \subset X \Rightarrow X \rightarrow Y$
- Augmentation : $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$
- Transitivité : $X \rightarrow Y$ et $Y \rightarrow Z \Rightarrow X \rightarrow Z$

Exemple : On a NUM_VOL → (NUM_AV, H_D, H_A)
et (NUM_AV, H_D, H_A) → (V_D, V_A)

donc NUM_VOL → (V_D, V_A)

- Union : $(X \rightarrow Y$ et $X \rightarrow Z) \Rightarrow X \rightarrow YZ$

Démonstration : si $X \rightarrow Y \Rightarrow XX \rightarrow XY$ (par augmentation) $\Rightarrow X \rightarrow XY$
et $X \rightarrow Z \Rightarrow XY \rightarrow ZY$ (par augmentation)
donc $X \rightarrow YZ$ (par transitivité)

- Pseudo transitivité : $(X \rightarrow Y$ et $WY \rightarrow Z) \Rightarrow WX \rightarrow Z$

Démonstration : si $X \rightarrow Y$ et $WX \rightarrow WY$ (par augmentation)
or $WY \rightarrow Z \Rightarrow WX \rightarrow Z$ (par transitivité)

- Décomposition : si $(X \rightarrow Y$ et $Z \subset Y) \Rightarrow X \rightarrow Z$

3.3.3. DEPENDANCES FONCTIONNELLES ELEMENTAIRES (DFE)

Une DFE est une dépendance $X \rightarrow A$, où A est un attribut unique non inclus dans X, et il n'existe pas de X' contenu dans X tel que X' détermine A.

Nota : La seule règle s'appliquant sur les DFE est la transitivité.

Exemples : (NUM_AV, H_D) → V_D est une DFE.

(NUM_AV, H_D,H_A) → V_D n'est pas une DFE.

3.4. FERMETURE TRANSITIVE ET COUVERTURE MINIMALE

3.4.1. FERMETURE TRANSITIVE

Ensemble de DFE augmenté des DFE obtenues par transitivité.

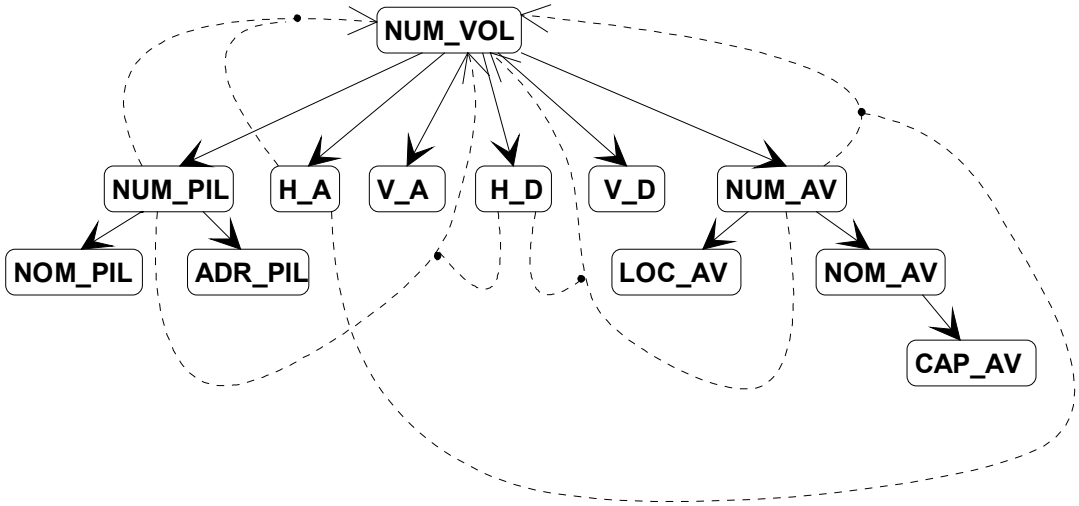
Exemple : F= {NUM_VOL → NUM_PIL, NUM_VOL → NUM_AV}

$F^+ = F \cup$ {NUM_VOL → NOM_PIL, NUM_VOL → ADR_PIL,
NUM_VOL → NOM_AV, NUM_VOL → CAP_AV, NUM_VOL → LOC_AV}

obtenu avec NUM_PIL → (NOM_PIL, ADR_PIL) et
NUM_AV → (NOM_AV, CAP_AV, LOC_AV)

Remarque : Deux ensembles de DFE sont équivalents s'ils ont la même fermeture transitive.

Graphe des dépendances fonctionnelles



3.4.2. COUVERTURE MINIMALE

C'est un ensemble F de DFE associé à un ensemble d'attributs vérifiant les propriétés :

- Aucune dépendance dans F n'est redondante.
  i.e. : pour toute DF f de F, F - {f} n'est pas équivalente à F

- Toute DFE des attributs est dans la fermeture transitive F^+ de F.

Remarque : On montre que tout ensemble de DFE, quelqu'il soit, a une couverture minimale qui n'est généralement pas unique.

Exemple : AVION(NUM_AV, NOM_AV, CAP_AV, LOC_AV)

- Ensemble des DFE et fermeture transitive :
  {NUM_AV → NOM_AV, NUM_AV → LOC_AV, NOM_AV → CAP_AV, NUM_AV → CAP_AV}

- Couverture minimale :
  {NUM_AV → NOM_AV, NUM_AV → LOC_AV, NOM_AV → CAP_AV}

3.5. NOTION DE CLE ET TROISIEME FORME NORMALE

3.5.1. CLE DE RELATION

Une clé de relation R(A1, ..., An) est un sous-ensemble X des attributs tel que :

- X → A1, A2, ..., An
- Il n'existe pas de sous-ensemble Y ⊂ X tel que Y → A1, A2, ..., An

Exemple : Dans la BDD AVION, NUM_VOL est une clé de la relation VOL.

Remarques 1 : Toute relation possède au moins une clé (A1, ..., An → A1, ..., An).
2 : Une relation peut avoir plusieurs clés :
- Clé primaire (primary key)
- Clés candidats

3.5.2. LES TROIS PREMIERES FORMES NORMALES

CODD : Décomposition des relations sans perte.

✪ Première forme normale

Une relation est en 1NF sit tout attribut est atomique.

Exemple : PERSONNE(AGE, ADRESSE) n'est pas en 1NF si ADRESSE(VILLE, RUE, NUMERO).

✪ Deuxième forme normale

Une relation est en 2NF ssi :

- La relation est en 1NF
- Tout attribut n'appartenant pas à une clé ne dépend pas que d'une partie de cette clé.

Exemples 1 : Clé primaire NUM_VOL alors la relation VOL est en 2NF.

2 : Clé (NUM_AV, H_D) alors la relation VOL n'est pas en 2NF
car NUM_AV → NOM_AV
→ CAP_AV
→ LOC_AV

Relation VOL décomposée en 2 relations :

AVION(NUM_AV, NOM_AV, LOC_AV, CAP_AV) en 2NF

VOL1(NUM_VOL, NUM_AV, NOM_PIL, NUM_PIL, ADR_PIL, H_D, H_A, V_D, V_A) en 2NF

✪ Troisième forme normale

Une relation est en 3NF ssi :

- La relation est en 2NF
- Tout attribut n'appartenant pas à la clé ne dépend pas d'un attribut non clé.

Exemples 1 : PILOTE(NUM_PIL, NOM_PIL, ADR_PIL) en 3NF.

2 : AVION(NUM_AV, NOM_AV, LOC_AV, CAP_AV) n'est pas en 3NF car NOM_AV → CAP_AV.

On décompose :

TYPE(NOM_AV, CAP_AV) en 3NF
AVION1(NUM_AV, NOM_AV, LOC_AV) en 3NF

3.5.3. PROPRIETES DES DECOMPOSITIONS EN 3NF

C'est une décomposition en {R1, R2, ..., Rp} d'une relation R qui préserve les DF est telle que la fermeture transitive des DF de R est la même que celle de l'union des DF de {R1, R2, ..., Rp}.

Exemple : Si l'on décompose maladroitement la relation AVION en
AVION1(NUM_AV, NOM_AV)
AVION2(NOM_AV, CAP_AV, LOC_AV)
Cette décomposition ne préserve pas les DF.
On perd NUM_AV → LOC_AV

Propriétés :
- La décomposition préserve les DF.
- La décomposition est sans perte.

Cette décomposition n'est pas forcément unique.

3.5.4. ALGORITHME DE DECOMPOSITION EN 3NF

Entrée : Schémas ne contenant que des DF.

Sortie : Schéma (R1, R2, ..., Rn) avec Ri en 3NF, quel que soit i.

Etape 1
Soit F l'ensemble des DF. Pour toute DF f, rendre f élémentaire.
Soit F' l'ensemble obtenu.

Etape 2
Rechercher une couverture minimale de F' notée MIN(F').

Etape 3
Partitionner MIN(F') en groupes F'1, F'2, ..., F'k tels que toutes les DF d'un même groupe aient la même partie gauche.

Etape 4
Pour chaque groupe F'i, i =1, ..., k, construire un schéma contenant les attributs de F'i et les DF de F'i.
Les éléments isolés (non déterminés) sont regroupés dans une relation dont ils constituent la clé.

Application de l'algorithme :

Etape 1
Toutes les DF sont élémentaires (voir graphe des DF précédent).

Etape 2 et 3

Couverture minimale :

NUM_VOL → NUM_AV	NUM_VOL → NUM_PIL	**Groupe F'1**
NUM_VOL → H_A	NUM_VOL → V_D	
NUM_VOL → H_D	NUM_VOL → V_A	
NUM_AV → NOM_AV	NUM_AV → LOC_AV	Groupe F'2
NOM_AV → CAP_AV		Groupe F'3
NUM_PIL → NOM_PIL	NUM_PIL → ADR_PIL	Groupe F'4

Etape 4
On a les relations en 3NF suivantes :

R1(NUM_VOL, NUM_AV, NUM_PIL, H_D, H_A, V_D, V_A)
R2(NUM_AV, NOM_AV, LOC_AV)
R3(NOM_AV, CAP_AV)
R4(NUM_PIL, NOM_PIL, ADR_PIL)

Les relations en 3NF comportent encore des redondances.

Autre exemple :
CODE_POSTAL(VILLE, RUE, CODE) en 3NF, avec la DFE CODE → VILLE.

CP	VILLE	RUE	CODE
	Paris	Arras	75005
	Paris	Jussieu	75008
	Paris	Monge	75005
	Paris	Le pic	75008

3.5.5. FORME NORMALE DE BOYCE-CODD

Une relation est en BCNF ssi les seules DFE sont celles dans lesquelles une clé détermine un attribut.

Propriétes :
- Toute relation a une décomposition en BCNF qui est sans perte.
- Par contre, toute décomposition en BCNF ne préserve pas les DF.

Exemple : CP(VILLE, RUE, CODE) avec la DFE CODE → VILLE
se décompose en :

VILLE_CODE(VILLE, CODE)

VILLE_CODE	VILLE	CODE
	Paris	75002
	Avignon	84000

RUE_CODE(RUE, CODE)

RUE_CODE	RUE	CODE
	Arras	75005
	Jussieu	75005
	Halles	75001
	Halles	84000
	Pasteur	84000

3.6. DEPENDANCES MULTI VALUEES ET 4NF

Soit la relation : ETUDIANT(NUM_ET, COURS, SPORT) en BNCF.

NUM_ET	COURS	SPORT
100	SGBD	Tennis
100	SGBD	Plongée
100	PROG	Tennis
100	PROG	Plongée

Soit R(A1, A2, ..., An) un schéma de relation et X et Y des sous-ensembles de {A1, ..., An}. On dit que X **multidétermine** Y (noté $X \rightarrow\rightarrow Y$) si étant donné des valeurs de X, il y a un ensemble de valeurs associées de Y, et cet ensemble est indépendant des autres attributs Z = R - X - Y de la relation R.
$(X \rightarrow\rightarrow Y) \Leftrightarrow \{ (xyz) \text{ et } (xy'z') \in R \Rightarrow (xy'z) \text{ et } (xyz') \in R\}$

Exemples : X = {NUM_ET} Y = {COURS} Z = {SPORT}
NUM_ET $\rightarrow\rightarrow$ COURS et NUM_ET $\rightarrow\rightarrow$ SPORT

(100 SGBD Tennis) et (100 PROG Golf)
$\Rightarrow$ (100 PROG Tennis) et (100 SGBD Golf)

3.6.1. PROPRIETES DES DEPENDANCES MULTI-VALUEES

✪ **Les Dépendances Fonctionnelles sont des cas particuliers des Dépendances Multi-valuées** :

Dém : si $X \rightarrow\rightarrow Y$, alors par def : xyz et x' y' z' $\in$ R
et y = y'
donc x y' z et xyz $\in$ R $\Rightarrow X \rightarrow\rightarrow Y$

✪ **La complémentation** : **si $X \rightarrow\rightarrow Y$, alors par def : $X \rightarrow\rightarrow R - X - Y$**

✪ **Augmentation** : **si $((X \rightarrow\rightarrow Y)$ et $V \subset W)) \Rightarrow XW \rightarrow\rightarrow YV$**

Dém : (x w x y v z) et (x w y' v' z') $\in$ R
or $V \subset W \Rightarrow v' = v$
$\Leftrightarrow$ (x w x y z) et (x w y' z') $\in$ R
or $X \rightarrow\rightarrow Y$, alors (x y w z') et (x y' w z) $\in$ R
$\Leftrightarrow XW \rightarrow\rightarrow YV$

✪ **Transitivité** : **$(X \rightarrow\rightarrow Y)$ et $(Y \rightarrow\rightarrow Z) \Rightarrow (X \rightarrow\rightarrow Z - Y)$**

Dém : si $Y \cap Z = \varnothing$ la transitivité est vérifiée.
sinon, la propriété se démontre...

✪ **Union** : **si $(X \rightarrow\rightarrow Y)$ et $(X \rightarrow\rightarrow Z) \Rightarrow X \rightarrow\rightarrow YZ$**
↓
(ensemble des attributs de Y et Z)

✓ **Une Dépendance Multi-valuée Elémentaire est une DM $(X \rightarrow\rightarrow Y)$ où** :

- Y n'est pas vide et Y est disjoint de X.
- Il n'existe pas de DM X'→→ Y' telle que X' ⊂ X et Y' ⊂ Y.

3.6.2. QUATRIEME FORME NORMALE : 4NF

Une relation est en 4NF ssi les seules DME sont celles dans lesquelles une clé (entière) détermine un attribut.

✓ Du fait qu'une DF est une DM, une relation en 4NF est en BCNF.

Exemple : ETUDIANT(NUM_ET, COURS, SPORT)

Cette relation est constituée des deux DME suivantes :

NUM_ET→→ COURS
NUM_ET→→ SPORT

ce ne sont pas des DME de la forme clé → attribut

Donc la relation n'est pas en 4NF, il faut la décomposer :

✓ On démontre que pour toute relation, il existe une décomposition (pas forcément unique) en relations en 4NF qui est sans perte.

Exemple de décomposition : ETUDIANT_COURS(NUM_ET, COURS) en 4NF
ETUDIANT_SPORT(NUM_ET, SPORT) en 4NF

3.6.3. ALGORITHME DE DECOMPOSITION EN 4NF

Entrée :Schéma R(X) contenant des DF et des DM.

Sortie : Schéma de plusieurs relations {R1, R2, ..., Rn} avec les Ri en 4NF.

ETAPE 1 (initialisation) : S = {R}

ETAPE 2 (itération) :

- Si T est un schéma de S qui n'est pas en 4NF, chercher une DM non triviale W→→V de T telle que W ne contienne pas une clé de T.
- Remplacer le schéma T dans S par deux schémas T1(W,V) et T2(X - V) munis des dépendances dérivées de la fermeture du schéma T.
- Répéter l'étape 2 tant qu'il existe dans S une relation qui n'est pas en 4NF.

ETAPE 3 (élimination de la redondance) : Pour tout couple (Ri, Rj) de S, si Xi ⊂ Xj, alors éliminer Ri de S.

3.7. DEPENDANCES DE JOINTURE ET 5NF

Les relations en 4NF comportent encore des redondances et des anomalies.

Exemple : VIN(BUVEUR, CRU, PRODUCTEUR)

VINS	BUVEUR	CRU	PRODUCTEUR
	Yves	Chablis	Claude
	Yves	Chablis	Nicolas
	Henri	Volnay	Nicolas
	Paul	Chablis	Nicolas

Cette relation est en 4NF mais comporte des redondances (ex: Nicolas produit du Chablis).

3.7.1. DEPENDANCES DE JOINTURE

Soit R(A1, A2, ..., An) un schéma de relation et X1, X2, ..., Xm des sous-ensembles de {A1, A2, ..., An}.

On dit qu'il existe une dépendance de jointure DJ notée : ⋇{XY, XZ} , si R est la jointure de ses projections sur X1, X2, ..., Xm.

$$R = \Pi X1(R) \boxtimes \Pi X2(R) \boxtimes \ ... \boxtimes \Pi Xm(R)$$

✓ Les DM sont des cas particuliers de DJ. En effet, R(X, Y, Z) vérifiant la DM X→→Y, alors X→→Z satisfait la jointure ⋇{XY, XZ}.

Exemple : La relation VINS peut se décomposer ainsi :
⋇{BUVEUR CRU, BUVEUR PRODUCTEUR, CRU PRODUCTEUR}

VINS	BUVEUR	CRU	PRODUCTEUR
	Yves	Chablis	Claude
	Yves	Chablis	Nicolas
	Henri	Volnay	Nicolas
	Paul	Chablis	Nicolas

R1	CRU	PRODUCTEUR
	Chablis	Claude
	Chablis	Nicolas
	Volnay	Nicolas

R2	BUVEUR	CRU
	Yves	Chablis
	Henri	Volnay
	Paul	Chablis

R3	BUVEUR	PRODUCTEUR
	Yves	Claude

Yves	Nicolas
Henri	Nicolas
Paul	Nicolas

Or R est la jointure de ses projections.

R1⊠R2	BUVEUR	CRU	PRODUCTEUR
	Yves	Chablis	Claude
	Yves	Chablis	Nicolas
	Henri	Volnay	Nicolas
	Paul	Chablis	Nicolas
	Paul	Chablis	Claude

R1⊠R2⊠R3	BUVEUR	CRU	PRODUCTEUR
	Yves	Chablis	Claude
	Yves	Chablis	Nicolas
	Henri	Volnay	Nicolas
	Paul	Chablis	Nicolas

3.7.2. CINQUIEME FORME NORMALE : 5NF

Les DJ sont induites par les clés candidates.

A titre d'exemple, soit R(A1, A2, A3, A4) , une relation ayant A1 et A2 comme clés candidates. Alors il est possible de décomposer la relation en :

$\ast${A1 A2, A1 A3, A1 A4} ou $\ast${A2 A1, A2 A3, A2 A4}

La connaissance des clés implique la connaissance de DJ.

PROCEDURE IMPLIQUE qui répond vrai ou faux à la question : est-ce qu'une DJ est impliquée par un ensemble de clés K d'une relation R(X) ?

Fonction IMPLIQUE(K, DJ)

DJ : $\ast$ {X1, X2, ..., Xm}

K est un ensemble de clés, avec K1 $\rightarrow$ X, K2 $\rightarrow$ X, ..., Kr$\rightarrow$X

soit S = {X1, ..., Xm}

Tant que il existe Ki, Y $\in$ S et Z $\in$ S tels que Ki $\subset$ Y $\cup$ Z

Faire
enlever Y et Z de S
ajouter Y $\cup$ Z dans S
Fin faire

Fin tant que

Si X $\in$ S **alors** IMPLIQUE := vrai;
sinon IMPLIQUE := faux;

Fin IMPLIQUE

Remarque: Une relation R est en 5NF si et seulement si toute DJ est impliquée par les clés candidates de R.

4. LANGAGES DE MANIPULATION DE DONNEES (LMD)

4.1. INTRODUCTION

Un LMD est constitué de commandes qui permettent :

- L'interrogation de la base de données.
- La modification de la base (insertion, mise à jour, ...).
- La programmation à partir d'un langage Hôte.

Il existe trois types de langages :

- Langages fondés sur l'algèbre relationnelle (SQL).
- Langages fondés sur le calcul des prédicats et le calcul de tuples (QUEL).
- Langages fondés sur le calcul des prédicats et le calcul relationnel de domaines (DRC).

4.2. L'ALGEBRE RELATIONNELLE

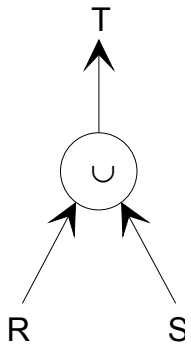
4.2.1. OPERATIONS DE BASE

Ces opérations binaires (union, différence, produit cartésien) ou unaires (projection, restriction) permettant de générer les autres (jointure, intersection, division).

4.2.1.1. UNION DE DEUX RELATIONS

L'union de deux relations de même schéma R et S est une relation T de même schéma contenant l'ensemble des tuples appartenant à R ou à S.

Notations: $T = R \cup S$
T = UNION (R,S)



Exemple:

VIN1	N°	CRU	AN	DEGRE
	100	Chablis	1976	13
	110	Médoc	1978	12
	120	Bourgogne	1977	12

VIN2	N°	CRU	AN	DEGRE
	100	Chablis	1976	13

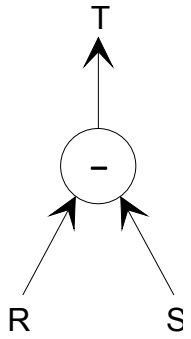
200	Beaujolais	1979	11

VIN1 ∪ VIN2	N°	CRU	AN	DEGRE
	100	Chablis	1976	13
	110	Médoc	1978	12
	120	Bourgogne	1977	12
	100	Chablis	1976	13
	120	Beaujolais	1979	11

4.2.1.2. DIFFERENCE DE DEUX RELATIONS

La différence R -S de deux relations R et S de même schéma est une relation T de même schéma contenant les tuples appartenant à R et n'appartenant pas à S.

Notations: T = R - S
T = MINUS (R,S)

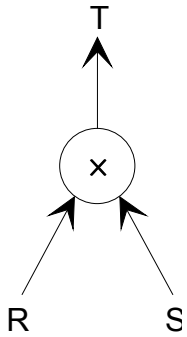


Exemple : VIN1 - VIN2

VIN1 - VIN2	N°	CRU	AN	DEGRE
	110	Médoc	1978	12
	120	Bourgogne	1977	12

4.2.1.3. PRODUIT CARTESIEN

Le produit cartésien de deux relations R et S de schémas quelconques est une relation T ayant pour attribut la concaténation des attributs de R et de S, et dont les tuples sont toutes les concaténations d'un tuple de R à un tuple de S.



Notations : T = R × S
T = TIMES(R, S)
T = PRODUCT(R, S)

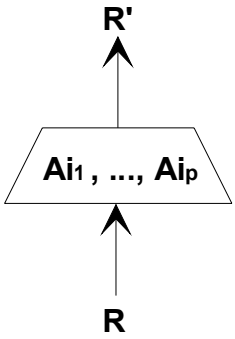
Exemple :

VIN3	N°	CRU	AN	DEGRE
	110	Médoc	1978	12
	120	Mâcon	1977	12

AGRICULTEUR	NOM	VILLE	REGION
	Paul	Marseille	B. du Rhône
	René	Avignon	Vaucluse

N°	CRU	AN	DEGRE	NOM	VILLE	REGION
110	Médoc	1978	12	Paul	Marseille	B.d.R
110	Médoc	1978	12	René	Avignon	Vaucluse
120	Mâcon	1977	12	Paul	Marseille	B.d.R
120	Mâcon	1977	12	René	Avignon	Vaucluse

4.2.1.4. PROJECTION



La projection d'une relation R(A1, A2, ..., An) sur les attributs Ai_1,...., Ai_p ($i_j \neq i_k$ et $p < n$) est une relation R'(Ai_1, Ai_2, ..., Ai_p) dont les tuples sont obtenus par élimination des valeurs de R n'appartenant pas à R' et par suppression des tuples en double.

Notations : Π Ai_1, Ai_2, ..., Ai_p (R) R(Ai_1, Ai_2, ..., Ai_p)
PROJECT(R/Ai_1, Ai_2, ..., Ai_p)

Exemple :

VINS	N°	CRU	AN	DEGRE
	110	Médoc	1978	12
	120	Mâcon	1977	12
	100	Chablis	1976	13
	110	Bourgogne	1977	12

(VINS)

AN	DEGRE
1978	12
1977	12
1976	13

Π AN, DEGRE (VINS)

4.2.1.5. RESTRICTION

Une formule de qualification atomique ou critère de sélection atomique est de la forme Ai θ C, Ai attribut, C constante, θ un opérateur parmi {< = > <= >= ≠}.

Exemple : DEGRE > 12
CRU = "Chablis"

La formule de qualification ou critère de sélection est construite à partir des qualifications atomiques et des connecteurs logiques ET ou OU (priorité ET > priorité OU).

Exemple : CRU = "Chablis" ET DEGRE < 12
(CRU = "Chablis" OU CRU = "Bourgogne") ET DEGRE < 12

La restriction de la relation R par la qualification Q est une relation R' de même schéma dont les tuples sont ceux de R qui satisfont Q.

Notations : σ_Q (R)
R [Q]
RESTRICT(R/Q)

Exemple : Q = AN = 1977

4.2.2. OPERATIONS ADDITIONNELLES

Définition: Une **qualification atomique multi-attributs** est du type Ai θ Aj.

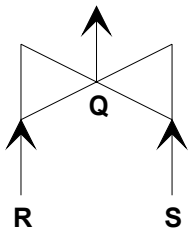
Exemple: CRU = VILLE

Une *qualification multi-attributs* est construite à partir des qualifications atomiques et des connecteurs logiques.

4.2.2.1. JOINTURE

La **jointure** de deux relations R et S selon une qualification Q est l'ensemble des tuples du produit cartésien R×S qui satisfont Q.

Notations: R⊠S
Q
JOIN [R,S]
R × S [Q]



Exemple: Jointure sur CRU = VILLE

VIN9	N°	CRU	ANNEE	DEGRE
	120	Mâcon	1978	12
	200	Saumur	1977	12
	210	Saumur	1979	14

VITICULTEUR	NOM	VILLE	REGION
	Paul	Tavel	Rhône
	Pierre	Mâcon	Bourgogne
	Jacques	Saumur	Loire

N°	CRU	ANNEE	DEGRE	NOM	VILLE	REGION
120	Mâcon	1978	12	Pierre	Mâcon	Bourgogne
200	Saumur	1977	12	Jacques	Saumur	Loire
210	Saumur	1979	14	Jacques	Saumur	Loire

Définitions:

- On parle d'**équijointure** de R et S avec Ai et Bj si (Ai = Bj).
- On parle de **θ jointure** de R et S avec Ai et Bj si (Ai θ Bj).
- L'**autojointure** de R selon Ai est la jointure de R avec elle-même selon (Ai = Ai).
- La **jointure naturelle** de R et S est l'équijointure de R et S selon tous les attributs de même nom suivie d'une projection (On ne trouve pas deux colonnes étant composées des mêmes éléments après jointure comme cela est le cas dans l'exemple précédent).
- La **semi-jointure** de la relation R par la relation Q est l'ensemble des tuples de R participant à la jointure de R et S selon Q.

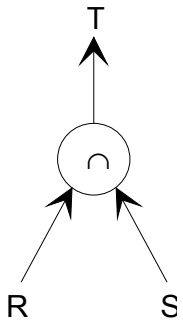
Notation: R •⊠ S

4.2.2.2. INTERSECTION

L'**intersection** de deux relations R et S de même schéma est une relation T de même schéma contenant les tuples appartenant à la fois à R et à S.

Notations: $R \cap S$
INTERSECT (R,S)

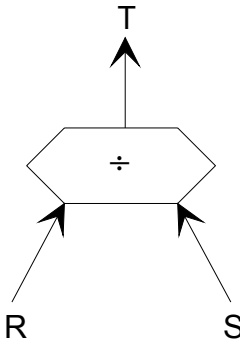
$R \cap S = R - (R - S)$
$R \cap S = S - (S - R)$



4.2.2.3. DIVISION

Le **quotient** de la relation R(A1, A2, . . ., An) par la relation S(A_{p+1}, . . ., An) est la relation T(A1, . . ., Ap) formée de tous les tuples qui concaténés avec chacun des tuples de S donne toujours un tuple de S.

Notation: $R \div S$
DIVISION (R,S)



- Si a_i est une valeur de l'attribut Ai alors T est définie par:

$T = \{ (a_1, \ldots, a_p) / \forall (a_{p+1}, \ldots, a_n) \in S, (a_1, \ldots, a_p, a_{p+1}, \ldots, a_n) \in R \}$

Exemple:

VIN10	CRU	ANNEE	DEGRE
	Mâcon	1977	12
	Mâcon	1979	14
	Mâcon	1980	12
	Saumur	1977	12
	Saumur	1979	14
	Chablis	1979	14

TYPE	ANNEE	DEGRE
	1977	12
	1979	14

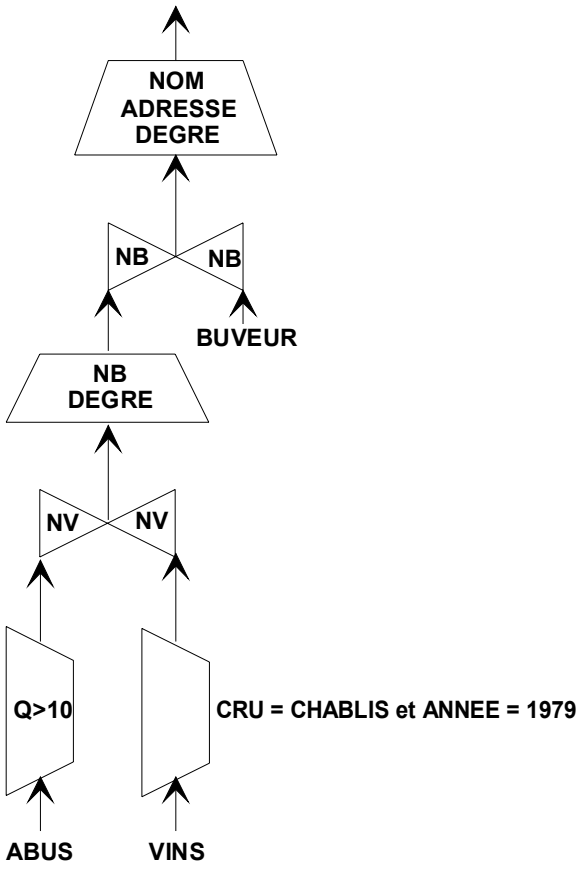
Quotient de VIN10 par TYPE:

CRU	CRU
	Mâcon
	Saumur

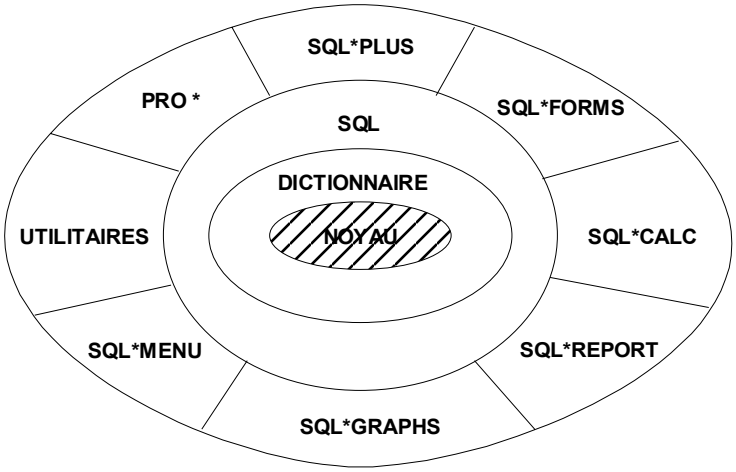
4.2.2.4. EXEMPLE D'APPLICATION

Soient les relations: VIN (NUM_VIN, CRU, ANNEE, DEGRE)
BUVEURS (NUM_BUV, NOM, ADRESSE)
ABUS (NUM_BUV, NUM_VIN, QUANTITE)

Problème: Quels sont les noms et adresses des buveurs ayant bu plus de dix bouteilles de Chablis 1979 et quel est le degré de ce vin ?



5. ORACLE: ARCHITECTURE FONCTIONNELLE



5.1. NOYAU

- Connexion avec d'autres noyaux de Bases de Données réparties
- Vérifications d'intégrité
- Vérification de la cohérence de données
- Contrôle des accès concurrents
- Gestion de la confidentialité des données
- Reprise sur panne
- Exécution optimale des requêtes
- Gestion des accélérateurs (index, cluster)
- Stockage physique des données

5.2. DICTIONNAIRE DES DONNEES

C'est une **méta base** qui décrit dynamiquement la Base de Données.

- Description des objets (tables, vues, index, séquences, synonymes, clusters, . . .)
- Utilisateurs, leurs droits, mots de passe, . . .

Utilisations du dictionnaire:

- Documentation
- Administration de la Base de Données

5.3. LA COUCHE SQL

LDD: Création, modification, suppression des structures de données.

LMD: Consultation, insertion, modification et suppression des données.

5.4. ARCHITECTURE REPARTIE D'ORACLE

Il y a deux types de répartition:

- Répartition des *applications*: Clients
  Serveur
  SQL*Net
- Répartition des *données*: Sites différents contenant des données.

6. ELEMENTS DE SQL

6.1. INTRODUCTION

6.1.1. NOTATIONS

[exp]	:	l'expression est optionnelle.
. . .	:	suite d'éléments du même type.
{ exp | exp | exp }	:	choix entre plusieurs expressions.
exp	:	valeur par défaut.

6.1.2. RELATIONS DE REFERENCE

AVION (NUM_AVION, NOM_AVION, CAPACITE, LOCALISATION)

PILOTE (NUM_PILOTE, NOM_PILOTE, ADRESSE)

VOL (NUM_VOL, NUM_AVION, NUM_PILOTE, VILLE_DEP, VILLE_ARR, HEURE_DEP, HEURE_ARR)

DUAL

SQL > SELECT sysdate FROM dual; {retourne la date système}
SQL > SELECT ascii ('Q') FROM dual; {retourne le code ascii du caractère Q}

6.2. OBJETS DE LA BASE

BASE DE DONNEES (DATABASE) : Fichier dans lesquels ORACLE mémorise les données.

TABLE (TABLE) : Structure de mémorisation des données.
Exemple: AVION, VOL, PILOTE.

COLONNE (COLUMN) : Attribut.
Exemple: capacité, ...

LIGNE (ROW) : Représente une entité unique.
Ordre des lignes indifférent.
Ordonner avec ORDER BY.

VALEUR (VALUE) : Donnée à l'intersection d'une ligne et d'une colonne.
Une valeur peut être nulle (i.e.: Il n'y a pas de valeur à cet endroit).

VUE (VIEW) : Représentation logique d'une table constituée soit par une autre table, soit par une combinaison d'autres tables (ou tables de base).

- Les données sont dérivées des tables de base

! C'est une table virtuelle!

- Les vues sont utilisées comme des tables.

- Si une vue est constituée d'une seule table, les opérations effectuées sur la vue affectent la table de base.

- Utilisation des vues:

  - Ajouter un niveau de sécurité pour limiter les opérations des utilisateurs.

  - Constituer des données complexes à partir de plusieurs tables.

  - Modifier les noms d'attributs.

INDEX (INDEX)

Permet d'accéder plus rapidement aux lignes ou de forcer l'unicité des lignes.

CLUSTER (CLUSTER)

Permet de structurer les données d'une ou de plusieurs tables en liant physiquement des groupes de lignes.

CLE PRIMAIRE (PRIMARY KEY)

Utilisée pour identifier de manière unique une ligne.

CLES ETRANGERES (FOREIGN KEYS)

Représentent des relations entre les tables.

CLE UNIQUE (UNIQUE KEY)

Clé primaire telle que :

- Une colonne définie comme clé unique doit avoir des valeurs différentes.

- Une clé unique ne peut pas avoir une valeur nulle.

- Il peut y avoir plusieurs clés uniques par table.

- Une clé unique peut comporter plusieurs colonnes.

CONTRAINTE (CONSTRAINT)

Les contraintes sont imposées soit à une colonne soit à une table entière.

SEQUENCE (SEQUENCE)

Objet utilisé pour générer des entiers.

SYNONYME (SYNONYM)

Nom donné à un objet qui permet de le référencier.

6.3. SYNTAXE DES NOMS D'OBJETS

1. Le nom d'un objet comporte au plus 30 caractères.
(8 pour les noms de la base)

2. Le nom peut contenir le caractère quote (**'**).

3. Le nom peut être écrit en majuscules ou en minuscules, c'est la même chose.

4. Un nom doit commencer par une lettre.

5. Un nom contient les caractères: lettres, chiffres, et **'**.

6. Un nom ne doit pas être un identificateur d'ORACLE.

7. Un nom ne doit pas être un identificateur pour un objet déjà utilisé.

8. Pour un objet qui est désigné au moyen d'un préfixe (ORAC1.VOL) on peut insérer des blancs autour du point (ORAC1 . VOL).

9. Un nom peut contenir n'importe quel caractère s'il est inclu entre guillemets (**"**).

Exemples: vol, nom_de_pilote, orac1 . avion, "Date de naissance"

6.4. TYPE DE DONNEES

6.4.1. TERMES

LITERAL (LITERAL) Valeur constante.
Chaque litéral a un type.

VARIABLE (VARIABLE) Désigne un litéral quelconque.

6.4.2. CARACTERES

CHAR ou **VARCHAR** Tout caractère imprimable.
Une colonne définie comme **CHAR** a au plus 255 caractères.

CHAR (n) Il y a n caractères dans la colonne.

6.4.3. TEXTES

Syntaxe: ' [caractères] . . . '

- Maximum : 255 caractères.
- Le caractère quote (') doit être dupliqué.

Exemples: 'oracle.dbo'
'aujourd''hui c''est Jeudi'

6.4.4. NOMBRES

6.4.4.1. DEFINITION DE COLONNES

Syntaxe: NUMBER [(précision [, décimal])]

- Si rien n'est indiqué, la longueur maximale est de 38 chiffres.
- "Précision" indique le nombre total de chiffres.
- "décimal" indique le nombre total de chiffres après la virgule.

Exemples:
```
SQL > CREATE  TABLE  table1
                (colonne1       NUMBER(5)
                (colonne2       NUMBER(15,2));
```

6.4.5. ECRITURE DES NOMBRES ENTIERS

Syntaxe: [+ / -] chiffre [chiffre . . .] [K | M]

- K multiplie par 1024.
- M multiplie par 1048576.
- Un entier est représenté sur 38 chiffres.

Exemples:
-255
29K
6M

6.4.5.1. NOMBRES DECIMAUX

Syntaxe: [+ / -] chiffre [chiffre . . .] • [chiffre . . .] [{ Exp | Mult }]

Mult [K | M]

Exp { e | E } [+ | -] chiffre [chiffre . . .]

Exemples:
634
7E2
25e-03
256K
-57.36e-12

6.4.5.2. DATES

Syntaxe: SYSDATE

- Centurie, année, mois, jour, heure, minute, seconde
- Le format standard pour l'heure est: 12:00:00 am
- Le format standard pour la date est: DD-MMM-YY
  05-SEP-91

6.4.5.3. CONVERSIONS ENTRE TYPES DE DONNEES

TO_char nombre → caractères
date → caractères

TO_number chaîne de chiffres → nombre

TO_date caractères → date
nombre → date

6.4.5.4. VALEURS NULLES

IS NULL
IS NOT NULL

Les conditions évaluées sur une valeur NULLE sont affectées à FALSE.

6.5. OPERATEURS

6.5.1. OPERATEURS ARITHMETIQUES

Opérateur	Fonction	Exemple
( )	forcer la priorité	select (x+y) / (x-y)
+ | –	opérateur unaire	where x < -2
∗ | /		where x > y / 2
+ | –	opérateur binaire	where y > x - 2

6.5.2. OPERATEURS SUR LES CHAINES DE CARACTERES

Opérateur	Fonction	Exemple
||	concaténation de chaînes (255 caractères maximum)	select 'nom' ||nom

6.5.3. OPERATEURS DE COMPARAISON

(résultat : TRUE, FALSE)

Opérateur	Fonction	Exemple
( )	force l'évaluation	not (a=1 or b=1)
=	égalité	where cap = 300
!= | ^= | <>	inégalité	where cap != 300
>	supérieur	where cap > 300
<	inférieur	where cap < 300
>=	supérieur ou égal	where cap >= 300
<=	inférieur ou égal	where cap <= 300
IN	appartenance à une liste ⇔ ANY	where x in ('A'...'Z')
NOT IN	non-appartenance à une liste ⇔ != ALL	where x not in (1,2)
ANY	comparer une valeur avec celles obtenues comme résultat d'une sous question	where loc = any (select loc from avion where cap>300)
ALL	comparer une valeur avec toutes celles d'une liste obtenues par une sous-question	where cap > all (select cap from avion where cap>300)
[NOT] BETWEEN X and Y	[non] compris entre X et Y	where x between 1 and 9

[NOT] EXISTS	TRUE s'il existe ou n'existe pas une ligne répondant à une question	where exists (select numpil from avion where adr='Pau')
[NOT] LIKE	[non] coïncidence avec la chaîne qui suit ("%" remplace toute chaîne "_" remplace 1 caractère)	where nomav like 'air %' (*airbus A300*) (*air___A300*)
IS [NOT] NULL	teste si une valeur est nulle ou non	where x is null

6.5.4. OPERATEURS LOGIQUES

( )	force l'évaluation	not (a = b and c = d)
NOT	négation	where not (x is null)
AND	conjonction	where x =1 and y = 1
OR	disjonction	where x = 1 or y = 1
UNION	regroupe toutes les lignes distinctes de 2 questions	select ... union select ...
INTERSECT	donne les lignes distinctes communes aux 2 questions	select.. intersect..select
MINUS	donnes les lignes distinctes de la 1ère question qui ne sont pas dans la 2ème	select ...minus select ...

6.5.5. OPERATEURS PARTICULIERS

Opérateur	**Fonction**	**Exemple**
*****	sélection de toutes les colonnes d'une table	select * from avion ;
COUNT(exp)	compte toutes les lignes où "exp" n'est pas nulle	select count(cap) from avion ;
COUNT(*)	compte toutes les lignes une seule fois	select count(*) from avion ;
DISTINCT	donne les lignes une seule fois	select distinct nomav from avion ;

6.6. FONCTIONS

6.6.1. FONCTIONS NUMERIQUES SUR UNE LIGNE

ABS	abs(n)	valeur absolue de n

Exemple : SQL> select abs(-15) "valeur absolue" from dual;
valeur absolue
15

CEIL	ceil(n)	plus petit entier inférieur ou égal à n
FLOOR	floor(n)	plus grand entier supérieur ou égal à n
MOD	mod(n,m)	n modulo m
POWER	power(n,m)	n^m
ROUND	round(n [,m])	arrondi avec 0 ou m chiffres après le point
SIGN	sign(n)	-1 si n < 0 ; 0 si n = 0 ; 1 si n > 0
SQRT	sqrt(n)	$\sqrt{n}$
TRUNC	trunc(n [,m])	n est tronqué avec 0 ou m décimales

6.6.2. FONCTIONS POUR LES CARACTERES SUR UNE LIGNE

CHR	chr(n)	caractères dont la valeur ASCII est n

Exemple : SQL> select chr(75) "caractère" from dual;
caractère
K

INITCAP	initcap(c)	écrit la chaîne avec une majuscule pour le 1er car.
LOWER	lower(c)	chaîne écrite en minuscules
LPAD	lpad(c_1 ,n [, c_2])	la chaîne c_1 est complétée à gauche jusqu'à la longueur n par des occurrences de c_2 (des blancs si c_2 est omis).
LTRIM	ltrim(c [,s])	supprime les occurrences de s de la partie gauche de c (blancs si s est omis).
RPAD	rpad(c_1 ,n [, c_2])	la chaîne c_1 est complétée à droite jusqu'à la longueur n par des occurrences de c_2 (des blancs si c_2 est omis).
RTRIM	rtrim(c [,s])	supprime les occurrences de s de la partie droite de c (blancs si s est omis).
SOUNDEX	soundex(c)	retourne des chaînes de caractères qui se prononcent de la même manière que c.

Exemple : SQL> select nompil from pilote where (nompil) = soundex('jake');
nompil

Jacques

SUBSTR	substr(c, m, [,n])	retourne la partie de la chaîne c commençant à partir du caractère à la position m et de longueur n (jusqu'à la fin si n est omis).
TRANSLATE	translate(c, s, r)	change dans c toutes les occurrences de s en r.
UPPER	upper(c)	la chaîne c est écrite en majuscules.

6.6.3. FONCTIONS QUI TRANSFORMENT UN CARACTERE EN VALEUR NUMERIQUE

ASCII	ascii(c)	renvoie la valeur ascii du caractère c.
INSTR	instr(c1,c2[,n[,m]]) omis).	position de la m^{ième} occurrence de c2 dans c1 commençant à la position n (1 si n et m
LENGTH	length(c)	longueur de c

6.6.4. FONCTIONS QUI REGROUPENT LES LIGNES DE RESULTAT

AVG	avg([distinct | all] col) nulles d'une colonne.	calcule la moyenne des valeurs non
COUNT	count([distinct | all] exp) pas nulle.	calcule le nombre de lignes où exp n'est
COUNT(*)		
MAX	max([distinct | all] exp) accessibles par exp.	calcule le maximum des valeurs
MIN	min([distinct | all] exp) accessibles par exp.	calcule le minimum des valeurs
STDDEV	stddev([distinct | all] col) la colonne.	écart type entre les valeurs non nulles de
SUM	sum([distinct | all] col)	somme entre les valeurs non nulles de la colonne.
VARIANCE	variance([distinct | all] col)	variance entre les valeurs non nulles de la colonne.

6.6.5. CONVERSION DES TYPES DE DONNEES

6.6.5.1. CONVERSION NOMBRE →CARACTERE

TO_CHAR to_char(n [,fmt])

Exemple : SQL> select to_char(17145,'$099,999') "caract" from dual;
caract
$017,145

TO_DATE to_date(c [,fmt])

Exemple : SQL> insert into indices (date_promotion) select
to_date('may 1,1991','month dd, yyyy) from dual;

6.6.5.2. CONVERSION CARACTERE →NOMBRE

TO_NUMBER to_number(c) séquence de chiffres → nombre

6.6.6. FONCTIONS POUR LES DATES

ADD_MONTH (d, n) ajoute les mois n à partir de la date d

LAST_DAY (d) retourne la date du dernier jour du mois de d

MONTH_BETWEEN (d, e) nombre de mois entre les dates d et e

NEXT_DAY (d, c) date du jour suivant c de la semaine d

SYSDATE date du jour

ROUND (d [, format]) date arrondie

Exemples :
```
SQL> SELECT ROUND(TO_DATE('28-apr-91'), 'year')
       "1er de l'an" from DUAL;
1er de l'an
01-JAN-92

SQL> SELECT TRUNC(TO_DATE('28-apr-91'),'year')
       "1er de l'an" from DUAL;
1er de l'an
01-JAN-91
```

GREATEST (d1, d2) date la plus récente de d1 et d2
LEAST (d1, d2) date la plus ancienne de d1 et d2

6.6.7. FONCTIONS UTILES

DECODE (exp, s1, r1 [s2, r2] ... [d]) Chaque fois que *exp* est égal à si, ri est retourné, sinon d est retourné.

Exemple :
```
SQL> SELECT nomav, DECODE(cap, 300, 'gros', 100, 'petit',
'moyen') from avion;

nomav  cap
atr    petit
b747   gros
DC9    moyen
```

NVL (e1, e2) si e1 est nul, NVL retourne e2, sinon e1

VID entier identifiant l'utilisateur

USER nom de l'utilisateur

VSIZE (exp) nombre d'octets pour représenter une expression ORACLE

6.6.8. FORMATS

6.6.8.1. NOMBRES

Caractère	Exemple	Description
9	9999 de la représentation.	9 est le nombre de caractères
0	0999	écrit des 0 avant le nombre.
$	$9999	préfixe la valeur avec $.
B	B999	écrit les 0 avec des blancs.
MI	9999MI	écrit "–" après les valeurs négatives.
PR	9999PR	valeurs négatives écrites entre "<" et ">".
,	9,999	
.	9.999	
V	999V99 de 9 après V.	multiplie par 10 fois le nombre
E	9.999EEEE	écrit en notation scientifique

6.6.8.2. DATES

CC ou **SCC** centurie, "S" préfixe avant J-C avec "."

YYYY ou **SYYY** année sur 4 chiffres (1993)

YEAR ou **SYEAR** année sur 3 caractères.

YYY ou **YY** ou **Y**

Q quart de l'année (1 : jan .. mars , 2 : avril .. juin ...)

MONTH mois sur 9 caractères

MON mois sur 3 caractères

MM mois en nombre (01 .. 12)

WW semaine de l'année (1 .. 52)

W semaine des mois

DDD jour de l'année (1 .. 366)

DD jour du mois (1 .. 31)

DAY nom du jour sur 9 caractères

DY abrégé du jour sur 3 caractères

J nombre de jours depuis le 1 jan 4712 B_C

AM / PM

HH24 heure du jour (1 .. 24)

HH ou **HHR** heure du jour (1 .. 12)

MI minutes (0 .. 59)

SS	secondes (0 .. 59)
.SSSS	secondes après minuit (1 .. 863999)
/. , .	ponctuation dans le résultat
" ... "	chaîne dans le résultat

7. PRINCIPALES INSTRUCTIONS SQL

7.1. CREATION D'OBJETS

7.1.1. CREATION D'UNE TABLE

CREATE TABLE [utilisateur.] table (colonne_def [,colonne_def] ...) [AS question]

avec colonne_def de la forme :

colonne *type de données* [DEFAULT exp] [*contrainte*]

- *colonne* : identificateur
- *type de données* : char | number | date
- DEFAULT exp : exp est la valeur par défaut de la colonne.
- *contrainte* : limite les variations des valeurs pour la colonne.

(définie par Oracle : SYS_C_n , n étant le n° de contrainte)

CONSTRAINT nom

NULL / **NOT NULL**
UNIQUE
CHECK condition Ex : check(dname = upper(dname))

Exemples :

```
CREATE TABLE
        Département
                ( Depno         number(2),
                  Depname       char(14),
                  Loc           char(13));

CREATE TABLE
        Employee
                ( Empno         number(4) not null,
                  Ename         char(10),
                  Job           char(9),
                  HGR           number(4),
                  Hiredate      date,
                  Sal           number(7,2) check (sal>40000) constraint
                        sal_min,
                  Comm          number(7,2),
                  depnonumber(2));
```

7.1.2. CREATION D'UNE VUE

CREATE VIEW [utilisateur.] vue AS question [WITH CHECK OPTION
[CONSTRAINT contrainte]]

Exemple :

```
create view bons_avions AS select * from Avion
            where numav NOT IN (select  numav from vol) with check option;
                                          ↓
                                      Contrainte                    |
                                                                    |
                                                                    ↓
```

(oblige à vérifier la contrainte si on entre un nouvel élément dans la vue).

7.1.3. CREATION D'UNE SEQUENCE

CREATE SEQUENCE [utilisateur.] séquence
[INCREMENT BY n1]
[START WITH n2]
[MAXVALUE n3 | NOMAXVALUE] *def* =10E27-1
[MINVALUE n4 | NOMINVALUE] def =-10E27-1
[CYCLE | NOCYCLE] *(souligné ⇔ pris par défaut)*
[CACHE n5 | NOCACHE]
[ORDER | NOORDER]

NEXTVAL : valeur suivante
CURRVAL : valeur courante

sequence.NEXTVAL

Exemples : CREATE SEQUENCE numpil
start with 1
increment by 1
nomaxvalue;

INSERT INTO pilote
VALUES (numpil.NEXTVAL, 'frantz', 'Munich', 35000);

7.1.4. CREATION D'UN SYNONYME

CREATE [PUBLIC] **SYNONYM** [utilisateur.] synonyme FOR [utilisateur.] objet

7.2. INSERTION DE DONNEES

INSERT INTO [utilisateur.] {table | vue} [(colonne [,colonne] ...)]
{VALUES (valeur [,valeur] ...) | question}

Exemples : INSERT INTO Dept VALUES (30, 'recherché', 'Avignon');

INSERT INTO Promotion (Ename, Job, Sal, Comm)
select Ename, Job, Sal, Comm from Employe
where Comm > 0.25*sal;

INSERT INTO vol_paris select numvol, numav, numpil, va, hd, ha
from vol where vd = 'Paris';

7.3. SELECTION DE DONNEES

SELECT [ALL | DISTINCT]
{* | [utilisateur.] {table | vue}.* | exp[c_alias]} *(c_alias = colonne alias)*
[,{utilisateur.] {table | vue}.* | exp[c_alias]}] ...
FROM [utilisateur.] {table | vue}.[t_alias]
[,[utilisateur.] {table | vue}.[t_alias]] ...
[WHERE condition]
[CONNECT BY condition] [START WITH condition]
[GROUP BY exp [,exp] ... [HAVING condition]]
[{UNION | INTERSECT | MINUS} question]
[ORDER BY {exp | position} [asc | desc] [,{exp | position} [asc | desc]] ...]

(position de la colonne à partir de laquelle s'effectue le tri}

Exemple : Quels sont les vols en correspondance directe ou indirecte au départ de Paris ?

SELECT numvol, vd, va from vol connect by vd = PRIOR va start with vd = 'Paris';

7.4. MODIFICATION DES DONNEES

7.4.1. MODIFICATION DES LIGNES DANS UNE TABLE

- **UPDATE** [utilisateur.] {table | vue} [alias]
  SET colonne = exp [colonne = exp] ...
  WHERE condition;

- **UPDATE** [utilisateur.] {table | vue} [alias]
  SET (colonne [, colonne] ...) = (question)
  [, (colonne [, colonne] ... = (question) ...]
  WHERE condition;

✓ - La clause WHERE est facultative.
- UPDATE permet de modifier une seule ligne, un ensemble de lignes ou toutes les lignes d'une table.

Exemple :
```
UPDATE avion
SET cap = cap +10
WHERE nomav = 'ATR';
```

7.4.2. EFFACEMENT DE LIGNES DANS UNE TABLE OU DANS UNE VUE

DELETE FROM [utilisateur.] {table | vue} [t_alias]
WHERE condition;

Exemple : DELETE FROM pilote where numpil = 5;

✓ Si la condition WHERE n'est pas précisée, toutes les lignes de la table seront effacées !

7.5. MODIFICATION DES OBJETS

7.5.1. MODIFICATION DES TABLES (INSERTION / SUPPRESSION DE COLONNES)

- AJOUTER UNE COLONNE

ALTER TABLE [utilisateur.] table
ADD (colonne colonne_def [, colonne colonne_def ...]);

Exemple : rajouter une colonne "*salaire*" à la table *PILOTE* (si *salaire* > 20000)

```
ALTER TABLE pilote ADD (sal NUMBER (7, 2) CHECK sal > 20000
                CONSTRAINT sal_num);
```

✓ - La valeur des champs rajoutés est nulle. Il n'est donc pas possible de préciser la condition "NOT NULL".
- La (ou les) colonne rajoutée est la dernière de la table.
- **Attention**, l'ajout d'une nouvelle colonne dans une table provoque une réorganisation (au moins partielle) de la base de données. Si une vue a été définie au moyen de colonnes faisant partie des modifications, elle doit être supprimée puis recréée.

- SUPPRIMER UNE COLONNE

ALTER TABLE [utilisateur.] table
DROP colonne;

ALTER TABLE [utilisateur.] table
DROP CONSTRAINT contrainte;

- MODIFIER LA LARGEUR D'UN CHAMP

ALTER TABLE [utilisateur.] table
MODIFY (colonne colonne_def [, colonne colonne_def ...]);

Exemple : agrandir la colonne *nompil* et supprimer la contrainte *sal_num*.

```
ALTER TABLE pilote
        MODIFY (nompil CHAR(20))
        DROP CONSTRAINT sal_num;
```

7.5.2. MODIFICATION DES SEQUENCES

ALTER SEQUENCE [utilisateur.] sequence
[INCREMENT BY n]
[START WITH n]
[MAXVALUE n | nom_maxvalue]
[MINVALUE n | nom_minvalue]
[CYCLE | NOCYCLE]
[CACHE n | NOCACHE]
[ORDER | NOORDER];

7.5.3. CHANGEMENT DE NOM D'UN OBJET

RENAME ancien_nom **TO** nouveau_nom;

! concerne les tables, vues et synonymes.

7.6. SUPPRESSION D'OBJETS

7.6.1. TABLES ET VUES

7.6.1.1. TABLES

DROP TABLE [utilisateur.] table;

! suppression possible si propriétaire en DBA.

supprime - toutes les lignes de la table
- tous les privilèges de la table
- invalide les vues et synonymes de la table

7.6.1.2. VUES

DROP VIEW [utilisateur.] vue;

7.6.2. SUPPRIMER DES SYNONYMES

DROP [PUBLIC] **SYNONYM** [utilisateur.] synonyme;

7.6.3. SUPPRIMER DES SEQUENCES

DROP SEQUENCE [utilisateur.] séquence;

On peut toujours revenir sur ce que l'on a fait :

7.7. VALIDATION DES COMMANDES

COMMIT [WORK]; *valide toutes les commandes que l'on a faites*

7.8. INVALIDATION DES OPÉRATIONS

ROLLBACK [WORK] [TO [SAVEPOINT] savepoint];

FIXER UN SAVEPOINT : **SAVEPOINT** savepoint;

8. SYNTAXE DES EXPRESSIONS

- FORME 1 : **colonne / constante ou valeur spéciale**

  [table.] colonne
  texte / nombre
  null | sysdate | md | user
  (sequence.CURVAL | sequence.NEXTVAL)

- FORME 2 : **fonction**

  nom_fonction([distinct | all] exp [, exp] ...)

  Exemples : length('Blake')
  round(1234.567*43)

- FORME 3 : **liste parenthésée d'expressions (priorité décroissante)**

Priorité décroissante

(exp)
+ exp | – exp | prior exp
exp*exp | exp/exp
exp+exp | exp–exp | exp | | exp

- FORME 4 : **combinaison d'expressions**

  (exp [, exp] ...)

9. SYNTAXE DES CONDITIONS

EVALUEES A **TRUE** OU **FALSE**

- FORME 1 : **comparaison avec une expression ou le résultat d'une question**

  <exp> <opérateur de comparaison> <exp>

  Exemple : ename = 'SMITH'

  <exp> <opérateur de comparaison> <exp_liste>
  <exp_liste> <égal ou différent> <question>

- FORME 2 : **comparaison avec l'un ou tous les membres d'une liste ou résultat d'une question**

  <exp> <op_comparaison> {any | all} (<exp> [, <exp>] ...)
  <exp> <op_comparaison> {any | all} <question>
  <exp_liste> <égal ou différent> {any | all} (<exp_liste>[, <exp_liste>] ...)
  <exp_liste> <égal ou différent> {any | all} <question>

- FORME 3 : **test d'appartenance à une liste sur une question**

  <exp> [NOT] in (<exp> [, <exp>] ...)
  <exp> [NOT] in <question>
  <exp_liste> [NOT] in (<exp_liste>[, <exp_liste>] ...)
  <exp_liste> [NOT] in <question>

- FORME 4 : **test d'inclusion**

  <exp> [NOT] between <exp> and <exp>

- FORME 5 :
  <exp> is [NOT] null

- FORME 6 :
  exists <question>

- FORME 7 : **combinaison d'autres conditions (priorité décroissante)**

  (<condition>)
  not <condition>
  <condition> and <condition>
  <condition> or <condition>

10. ADMINISTRATION

10.1. CREATION D'OBJETS

10.1.1. CREATION DE LA BASE

CREATE DATABASE [base] (base [8 car. max.] = ORACLE si non spécifié)
[CONTROL FILE REUSE]
[LOGFILE fichier_spec [, fichier_spec] ...]
[MAXLOGFILE entier]
[MAXDATAFILE entier]
[MAXINSTANCE entier]
[ARCHIVELOG | NOARCHIVELOG]
[EXCLUSIVE]

avec "fichier_spec" : nom [SIZE entier [K | M]] [REUSE] →(*les fichiers peuvent être réutilisés*)

✓ Dans le fichier INIT.ORA créé lors de la création de la base, se trouvent les noms des fichiers LOGFILE.

10.1.2. CREATION DE LA TABLESPACE (espace de rangement des données)

CREATE TABLESPACE nom
DATAFILE fichier_spec [, fichier_spec] ...
[DEFAULT STORAGE ([INITIAL n] [NEXT n] [MINEXTENTS n] [MAXEXTENTS n]
[PCINCREASE n])
[ONLINE | OFFLINE]
avec INITIAL par défaut à 1024 octets, MIN à 4096 octets et MAX à 4095 MO.

10.2. MODIFICATION DE LA BASE

10.2.1. MODIFICATION DE L'ORGANISATION LOGIQUE

ALTER DATABASE [base]
{ADD LOGFILE fichier_spec [, fichier_spec] ...
| DROP LOGFILE fichier_spec [, fichier_spec] ...
| RENAME LOGFILE fichier_spec [, fichier_spec] ...
TO fichier_spec [, fichier_spec] ...
| ARCHIVELOG | NOARCHIVELOG
| MOUNT[[EXCLUSIVE] | SHARED] | DISMOUNT
| OPEN | CLOSE [NORMAL | IMMEDIATE]}

10.2.2. MODIFICATION DE LA TABLESPACE

ALTER TABLESPACE nom
{ADD DATAFILE fichier_spec [, fichier_spec] ...
| RENAME DATAFILE fichier_spec [, fichier_spec] ...
TO fichier_spec [, fichier_spec] ...
| DEFAULT STORAGE([INITIAL n] [NEXT n] [MINEXTENTS n]
[MAXEXTENTS n] [PCINCREASE n])
| ONLINE | OFFLINE[NORMAL | IMMEDIATE]
| {BEGIN | END} BACKUP}

10.2.3. MODIFICATION DES UTILISATEURS (uniquement en DBA)

ALTER USER nom [IDENTIFIED BY mot_de_passe]
[DEFAULT TABLESPACE tablespace]
[TEMPORARY TABLESPACE tablespace]

10.3. PRIVILEGES

10.3.1. AUTORISATION D'ACCES A LA BASE

GRANT db_privilège [, db_privilège] ... **TO** utilisateur [, utilisateur] ...
[IDENTIFIED BY mot_de_passe [, mot_de_passe] ...]

avec db_privilège : privilège DBA ou
possibilité de se connecter (CONNECT) ou
RESOURCE (créer des tables dans la base)

10.3.2. AUTORISATION D'ACCES A LA TABLESPACE

GRANT RESOURCE [(quota [K | M)] **ON** tablespace **TO** {PUBLIC | utilisateur}
[, utilisateur] ...

10.3.3. ACCES AUX OBJETS

GRANT {objet_priv [, objet_priv] ... | ALL [PRIVILEGES]}
ON [utilisateur.] objet
TO {PUBLIC | utilisateur} [, utilisateur] ... [WITH GRANT OPTION]

10.3.4. SUPPRESSION D'ACCES A LA BASE (uniquement en DBA)

REVOKE {CONNECT [, RESOURCE] [, DBA]} **FROM** utilisateur [, utilisateur] ...

10.3.5. SUPPRESSION D'ACCES A UNE TABLESPACE

REVOKE RESOURCE ON tablespace **FROM** utilisateur [, utilisateur] ...

10.3.6. SUPPRESSION D'ACCES AUX OBJETS (pour les propriétaires de ces objets ou ceux ayant des privilèges sur ces objets)

REVOKE {objet_priv [, objet_priv] ... | ALL [PRIVILEGE]}
ON [utilisateur.] objet **FROM** {PUBLIC | utilisateur} [, utilisateur] ...

11. SQL*PLUS - COMMANDES INTERACTIVES

11.1. DEFINITION DE VARIABLES

DEF[INE] [variable | variable = texte]

✓ On peut définir 240 variables maximum

SQL>DEF : liste les variables existantes

[& : select &col ... from &table]

11.2. EXECUTION DES COMMANDES AVEC PARAMETRES

STA[RT] fichier [.ext] [arg1 arg2 ...]

Exemples : SELECT * from vol where vol = '&1' and hd <&2;

vols au départ de Paris après 10h : SQL>STA vol_vd_hd Paris 10;

11.3. ECRITURE DES MESSAGES A L'ECRAN

PROMPT [texte]

11.4. AFFECTATION DE VARIABLES PAR LECTURE

ACC[EPT] variable [NUM[BER] | CHAR]
[PROMPT texte | NOPR[OMPT]] [HIDE] →(*supprime l'affichage lors de l'entrée*)

Exemple : SQL>ACC salaire NUM PROMPT 'salaire : '

11.5. FORMATAGE DES RESULTATS

11.5.1. FORMATAGE DES COLONNES

COL[UMN] [{column | exp} [options] ...]

OPTIONS LES PLUS UTILISEES :

- **CLE**[AR] / **DEF**[AULT] remet les valeurs par défaut
- **FOLD_A**[FTER] n : insère un RC après la tête de col. et chaque lig. de la col.
- **FOLD_B**[EFORE] n : insère un retour chariot avant la tête de colonne et avant chaque ligne de la colonne.
- **FOR**[MAT] format

- **HEA**[DING] texte

  'capacité | avion' → capacité (*fait l'écriture sur deux lignes*)
  avion

- **JUS**[TIFY] {L[EFT] | C[ENTER] | R[IGHT]}
- **NEWL**[INE] : passe à la ligne avant d'écrire la valeur de la colonne.
- **NOPRI**[NT] | **PRI**[NT]
- **NUL**[L] chaîne_car : affecte NULL à quelque chose, ou affecte des blancs.
- **ON** | **OFF** : contrôle le statut de sortie des attributs de la colonne.

11.5.2. RUPTURES DE SEQUENCES DANS UN RAPPORT

BREAK [ON rapport_élément [[action] [action]]]

rapport élément : {colonne | exp | ROW | REPORT}

action : [SKI[P] n | SKI[P] PAGE]
[NODUP[LICATES] | DUP[LICATES]]

Action faite : - chaque fois qu'un élément de "colonne" change de valeur.
- chaque fois qu'un élément de "exp" (valeur) change de valeur.

11.5.3. TITRE D'UN RAPPORT

- **BTI**[TLE] [spe_impression [texte | variable] ...] | [OFF | ON] : Bas de page
- **TTI**[TLE] [spe_impression [texte | variable] ...] | [OFF | ON] : Haut de page

avec
spe_impression :
∗ COL n : positionne le titre à colonne n
∗ S[KIP] [n] : saute n lignes après le titre (n = 0 : même ligne).
∗ TAB n : positionne n colonnes - plus loin si n >0
- moins loin si n <0
∗ LE[FT] | CE[NTER] | R[IGHT]
∗ BOLD
∗ FORMAT c

texte : texte du titre

variable : SQL.LNO : N° de ligne courante
SQL.PNO : N° de page courante

11.5.4. IMPRESSION DE CALCUL SUR LES DONNEES

COMP[UTE] [fonction ...] **OF** {exp | colonne} **ON** {exp | colonne | REPORT | ROW}

fonction :
- AVG (nombre)
- COU[NT] pour compter les lignes ayant une valeur ≠ NULL
- MAX[IMUM] nombre, char
- NUM[BER] pour compter toutes les lignes
- STD
- SUM
- VAR[IANCE]

11.5.5. SUPPRESSION DES OPTIONS

CL[EAR] option

option :
- BRE[AKS]
- BUFF[ER]
- CO[LUMNS]
- COMP[UTES]
- SCR[EEN]
- SQL (vide le buffer : CLEAR SQL)

11.6. VARIABLES DE L'ENVIRONNEMENT

11.6.1. AFFECTATION D'UNE VARIABLE

SET variable_système valeur

avec variable système :

- ECHO {OFF | ON}
- HEA[DING] {OFF | ON}
- LIN[ESIDE] {80 | n} 1≤n≤500
- NEWP[AGE] {1 | n} n : nombre de lignes entre le haut et le titre.
- NULL texte
- NUMF[ORMAT] format format : format par défaut pour les nombres.
- NUM[WIDTH] {10 | n}
- PAGES[IZE] {54 | n} n : nb de lignes entre le haut & bas de page.
- PAU[SE] {OFF | ON | texte} s'arrête après chaque impression d'une page en affichant *texte*
- SPA[CE] {1 | n} n≤10 n : nombre de blancs entre les colonnes.

11.6.2. VISUALISATION DES VARIABLES

SHO[W] option SHOW ALL : toutes les variables

variables système :
- BTI[TLE]
- TTI
- LNO
- PNO
- REL[EASE]
- SPOO[L]
- USER

11.6.3. IMPRESSION DE MESSAGES (pauses)

PAU[SE] texte

11.6.4. SORTIE DES RESULTATS

SPO[OL] [fichier[. extension] | OFF | OUT]

LST par défaut OFF : arrête la sortie sur le spool OUT : et imprime

11.7. INFORMATIONS RELATIVES AUX OBJETS

11.7.1. DESCRIPTION DES TABLES

DESC[RIBE] [utilisateur.] objet

11.7.2. ACCES AU DICTIONNAIRE

11.7.2.1. DICTIONNAIRE

SQL>select * from **dict**;

11.7.2.2. UTILISATEURS

SQL>select * from **all_users**;

11.7.3. OBJETS PROPRIETAIRES

SQL>select * from **cat**;

tabs	: tables
cat	: no tables
syn	: no synonymes
seq	: séquences
obj	: objets
ind	: indexes
cols	: colonnes

11.7.4. OBJETS ACCESSIBLES

all_tables : toutes tables accessibles
accessibles_tables : toutes tables et vues accessibles
accessibles_columns

11.7.5. PRIVILEGES

SQL>select * from myprior;

$$\begin{cases} 0 \text{ si non} \\ 1 \text{ si oui} \end{cases}$$