## Modèle relationnel :

Un schéma de relation R, dénoté R(A1:D1, A2:D2, ..., An:Dn) est un ensemble d’attributs.

Chaque attribut Ai est le nom d’un rôle joué par son domaine Di dans le schéma de relation R. Une schéma de relation R est utilisé pour décrire une relation.

Une relation R correspond au sous ensemble du produit cartésien de n domaines : c = inclut

R c D1 × D2 × D3 × … × Dn (avec n le degré de la relation.)

Exemple : PILOTE : NUM\_PILOTE × NOM\_PILOTE ×ADRESSE × SALAIRE est la relation

Pilote (NumPil, NomPil, adr, sal)

Définir un schéma de relation revient à spécifier un nouveau type de données équivalent à un type STRUCT en C par exemple. Mais le modèle relationnel n’autorise qu’un seul niveau de structure : Il n'est pas possible par exemple de définir l’attribut Adresse qui se décompose en Rue, Ville et Code Postal.

Une relation r dénotée r(R) du schéma de relation R(A1:D1, A2:D2, ..., An:Dn) est un ensemble d'enregistrements

Degré = nombre d’attributs

Cardinalité = nombre de tuples

Une clé de relation est un sous-ensemble d'attributs qui permet de caractériser tout enregistrement d'une relation

• Une clé est invariante dans le temps.

• En général, il existe plusieurs clés pour une même relation R

• Parmi les clés possibles, on choisit une clé qui sera appelée clé primaire, les autres seront candidates

• Lors de la définition d'un schéma cette clé est mise en évidence (soulignement ou gras)

Domaine primaire : domaine de définition d’un attribut clé primaire

• 2 types de relations :

* Relations statiques (pas de Clé étrangère, indépendantes des autres)
* Relations dynamiques (avec Clé étrangère)

Avion (**NumAv**, AvNom, loc, cap) **relation statique**

Vol (**NumVol**, #NumPil, #NumAv, Vil\_dep, Vil\_arr, Heu\_dep, Heu\_arr) **relation dynamique**

Un schéma de base de données relationnel S est un ensemble de schémas de relation

S = {R1, R2, …, Rp} et un ensemble de **contraintes d'intégrité** CI

Une instance de base de données relationnelle BD est un ensemble d'instances de relations BD = {r1, r2, ..., rn} où chaque ri respecte les contraintes d'intégrité

Une **contrainte d’intégrité** est une propriété du schéma, invariante dans le temps.

• Il existe différents types de contraintes d'intégrité :

* Structurelles ou statiques (liées au modèle relationnel)
* Applicatives ou dynamiques (contraintes de cohérences liées à l’application)

Les contraintes du modèle :

**• CI de domaine**

* «toute valeur d’un attribut doit appartenir à son domaine de définition»

**• CI de relation**

* «toute valeur de clé primaire existe et est unique»

**• CI de référence**

* «Toute valeur de CE existe dans la CP associée»
* la valeur d'attribut de la relation r1 doit apparaître comme valeur de clé dans une autre relation r2

## Le langage algébrique :

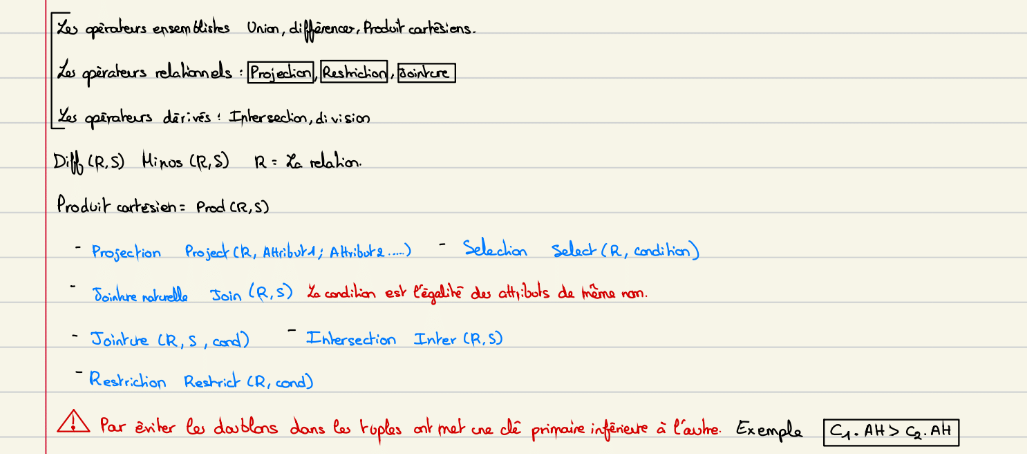
Les opérateurs relationnels :

• Deux catégories

* unaires de restriction
* binaires d’extension

• Les opérateurs de restrictions permettent :

* soit un découpage horizontal d’une relation (SELECTION)
* soit un découpage vertical d’une relation (PROJECTION)



## 

## Le modèle entité-association :

Modèle conceptuel (UML) -> modèle logique (Relationnel) -> modèle physique. (SQL)

• Association : représentation d'un lien non orienté entre plusieurs entités (qui jouent un rôle déterminé)

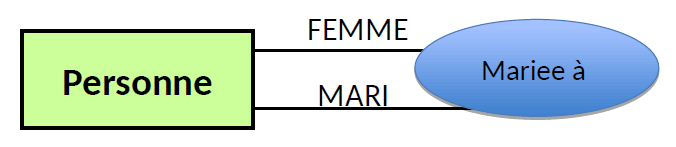
• Type d'association (TA) ou classe d’association : représentation d'un ensemble d'associations ayant la même sémantique et décrites par les mêmes caractéristiques



« 1 personne possède 1 maison »

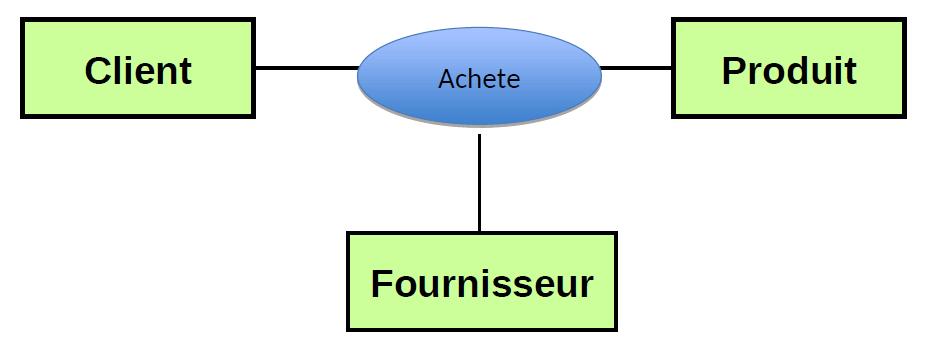
« 1 maison est achetée par une personne »

Association cyclique :

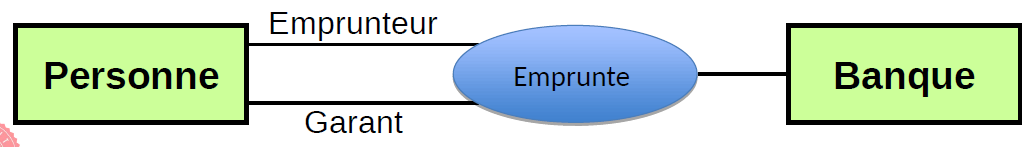


« mariée à » = < 1 personne/FEMME, 1 personne/MARI >

Associations ternaires :

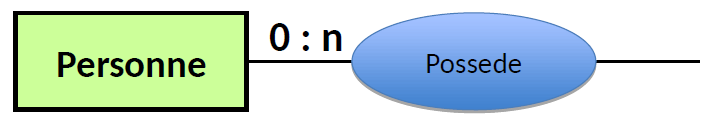


’achète’ = < 1 client, 1 produit, 1 fournisseur >

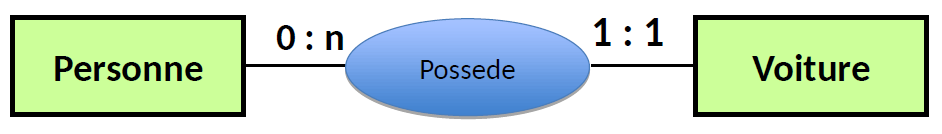


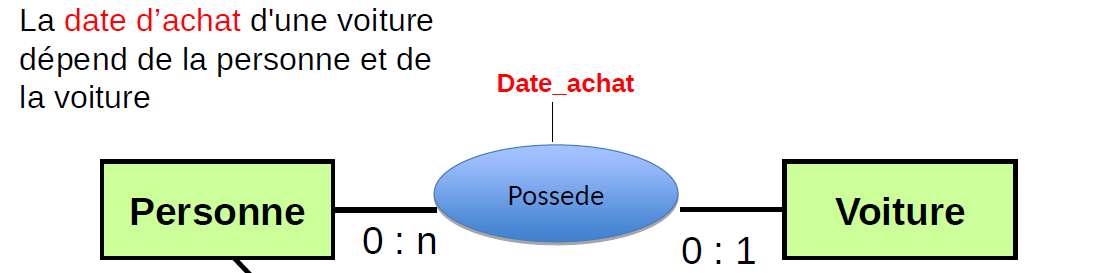
Cardinalité :

• Une personne peut **ne pas avoir** de voiture, en avoir 1, 2,… **n** (pas de contraintes)



• Une voiture peut avoir un et **un seul** propriétaire

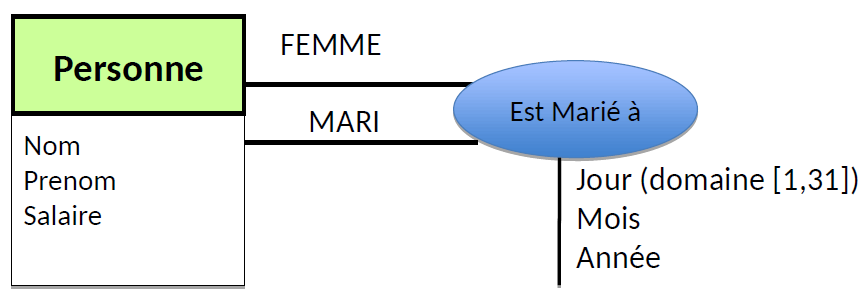




**Les attributs :**

• Décrivent les propriétés associées à :

* un type d’entité
* un type d’association
* un autre attribut (Attention)



• simple (atomique) : non décomposable

* Le domaine de valeurs est constitué de valeurs atomiques
* Ex. : jour - domaine de valeurs : {1, 2, …., 31}
* Domaines prédéfinis standard, intervalles, énumérés

• Complexe : décomposé en d'autres attributs

* Exemples: date (jour, mois, année), adresse (rue, ville, code postal)

• Pour chaque type d’entité il faut un identifiant qui permet de repérer une entité de manière unique et sans ambiguïté. Il peut être composé de plusieurs attributs (exemple numpilote, ou numexemplaire etc.)

**Les contraintes :**

• Monovalué (1 seule valeur) : Date de naissance

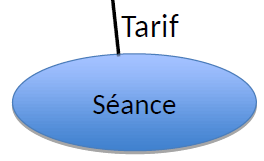
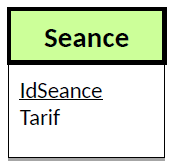
• Multivalué (plusieurs valeurs) (Attention) : Numéros de téléphones, prénoms

• Obligatoire : Nom (NOT NULL)

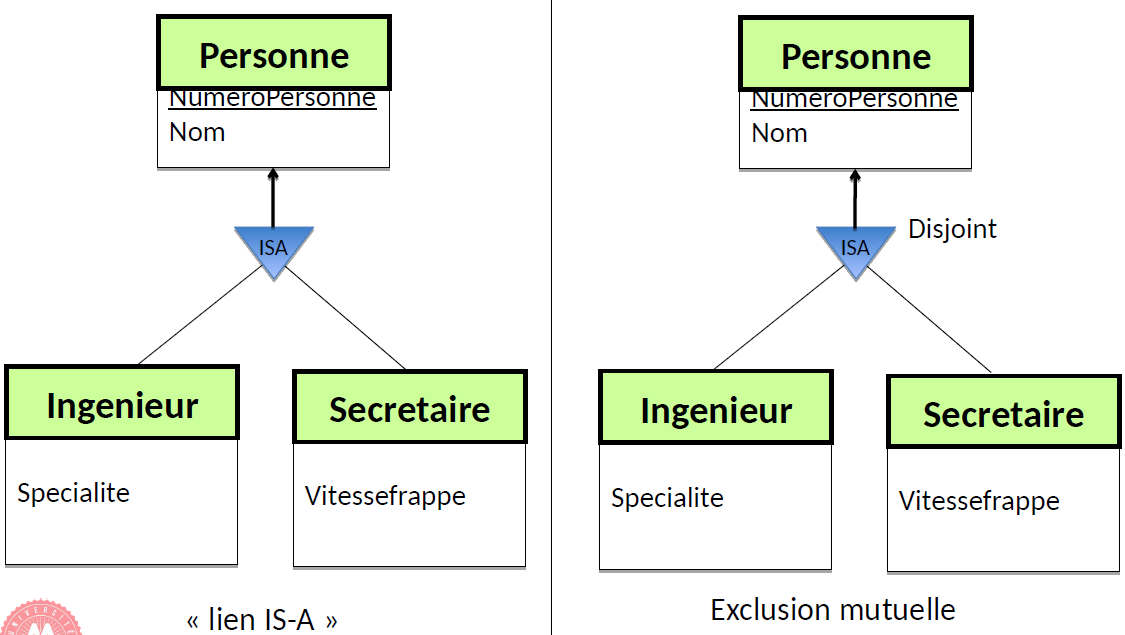
• Facultatif : Téléphone

Les relations ternaires sont généralement de type 1:N.

Transformation type d’association vers entité :

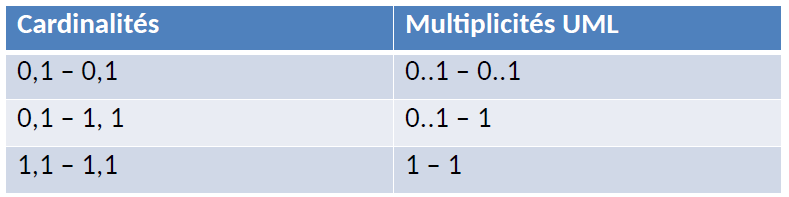
 devient 

• Transformer les relations ternaires peut mettre en évidence des problèmes de conception

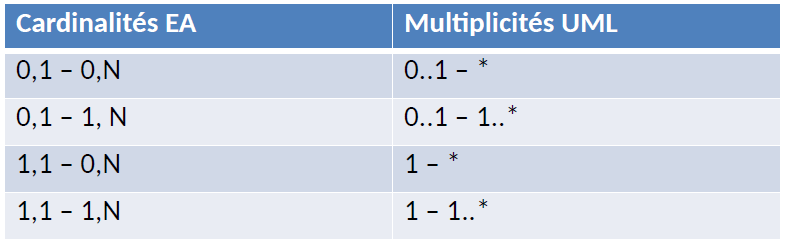


**Formalisme UML :**

• Associations 1 – 1 :



• Associations 1 - N :



ATTENTION : La notation de la cardinalité en UML est opposée à celle d’E/A.

UML : à gauche (resp. à droite) le nombre d'instances de la classe de gauche

(resp. de droite) autorisées dans l'association.

E/A : à gauche (resp. à droite) le nombre d'instances de la classe de droite

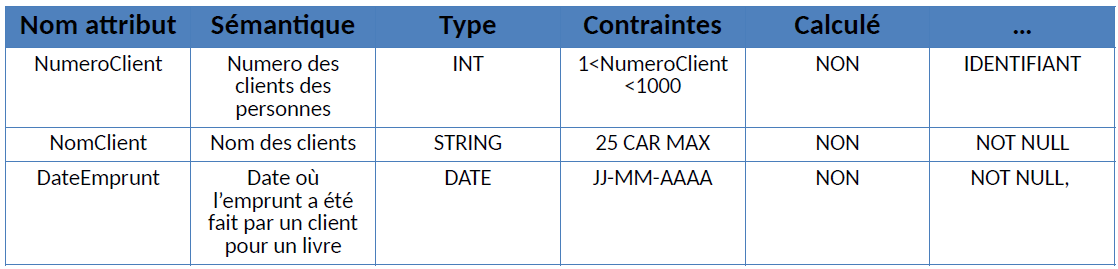
(resp. de gauche) autorisées dans l'association.

## 

• Le dictionnaire de données est important.

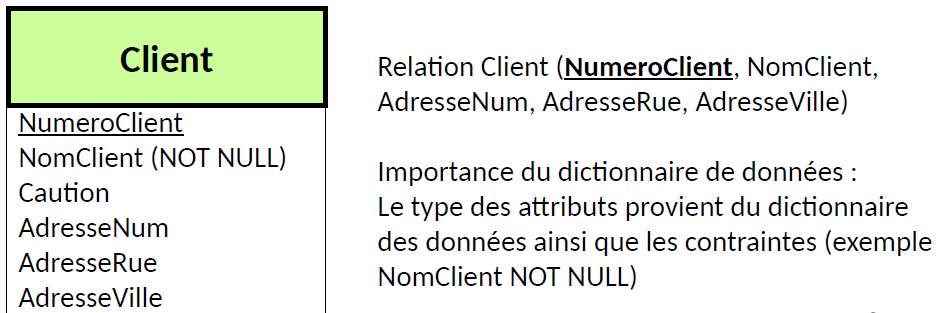
• Votre modélisation doit être capable de répondre aux différentes requêtes qui vont être posées par le suite. Pour s'entraîner : cours E/A diapo 65

Exemple d’un dictionnaire de données :



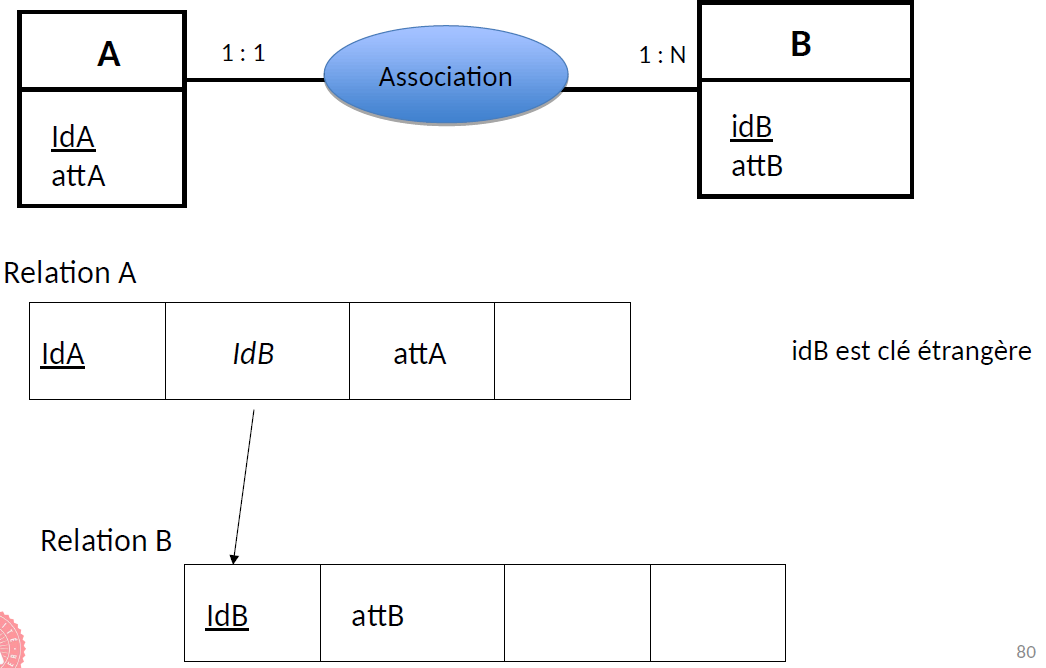
• Attention, ne pas oublier les contraintes, les reporter dans le dictionnaire de données pour certaines d’entre elles. Autrement les lister, elles serviront par la suite.

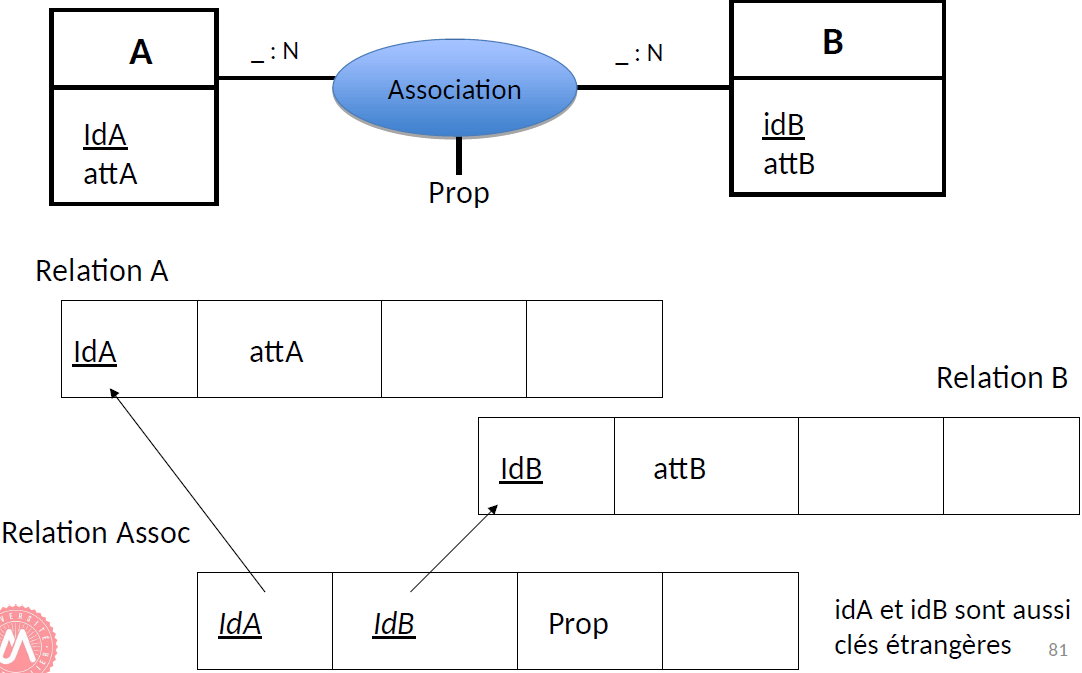
**Transformation E/A - Relationnel :**

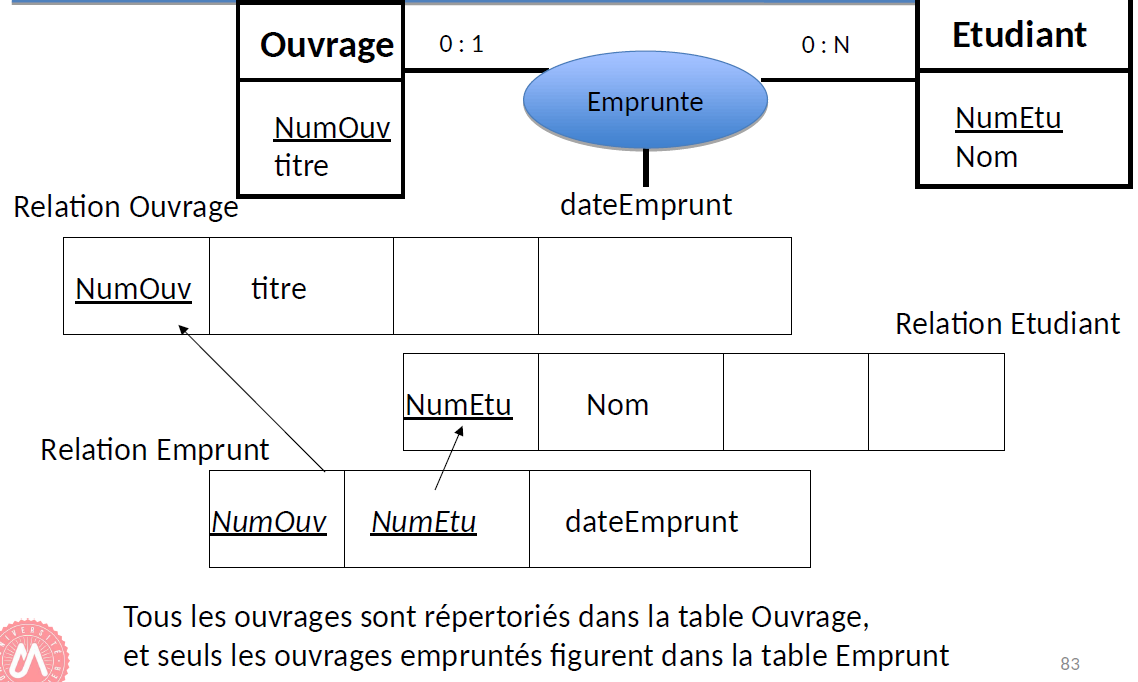


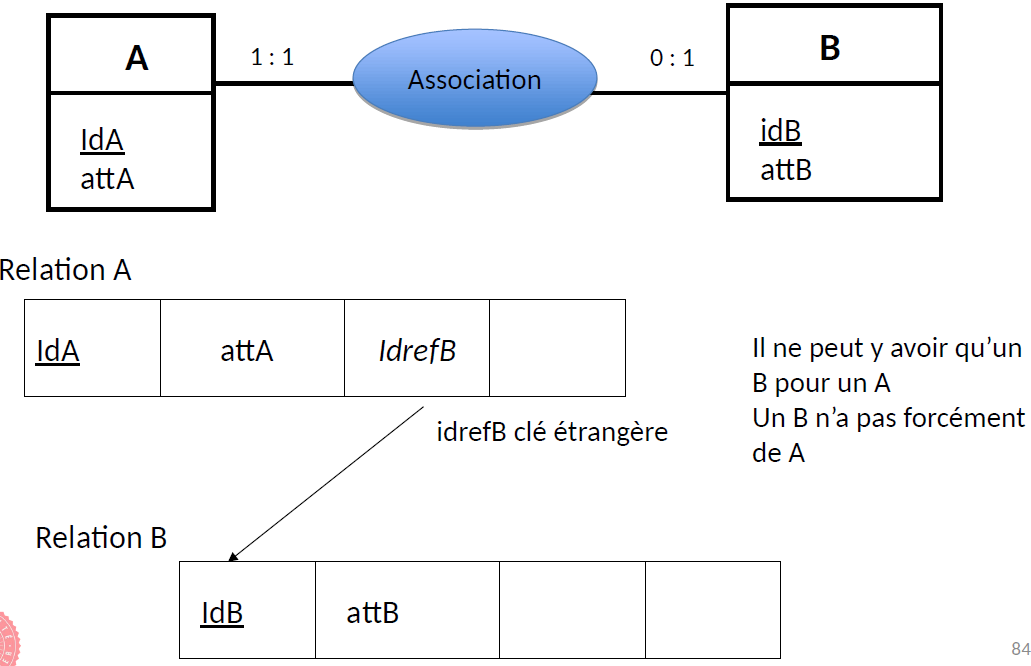
**Transformations pour les Types d’Associations**

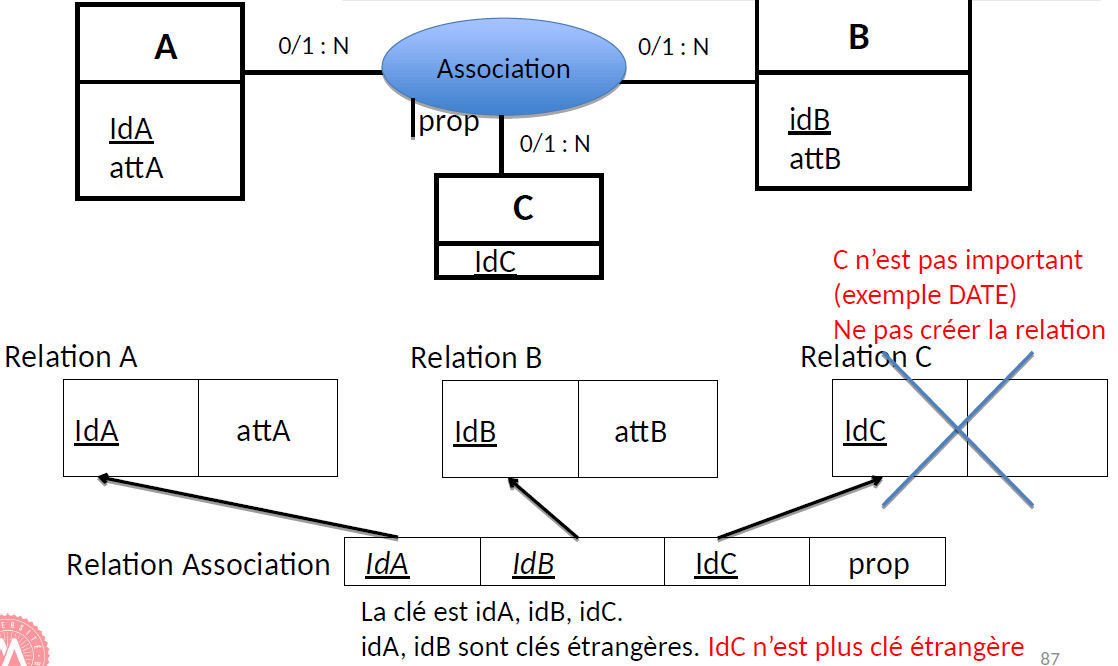
* cas 1 : cardinalité 1,1 sur l'une des pattes : compléter la relation concrétisant l'entité avec la patte 1,1 en y ajoutant une propriété qui référence l'identifiant de l'autre entité
* cas 2 : toutes les cardinalités maximale = n : créer une relation pour l'association, dont la clé se compose des clés des entités liées, avec éventuellement les propriétés portées par l'association
* cas 3 : cardinalité 0,1 sur l'une des pattes : choisir entre une traduction selon le cas 1 ou le cas 2 en veillant toutefois : dans le cas 1, si l’association a des propriétés, ne pas oublier de les adjoindre à la relation correspondant à l'entité avec la patte 0,1 et dans le cas 2 de simplifier l'identifiant.

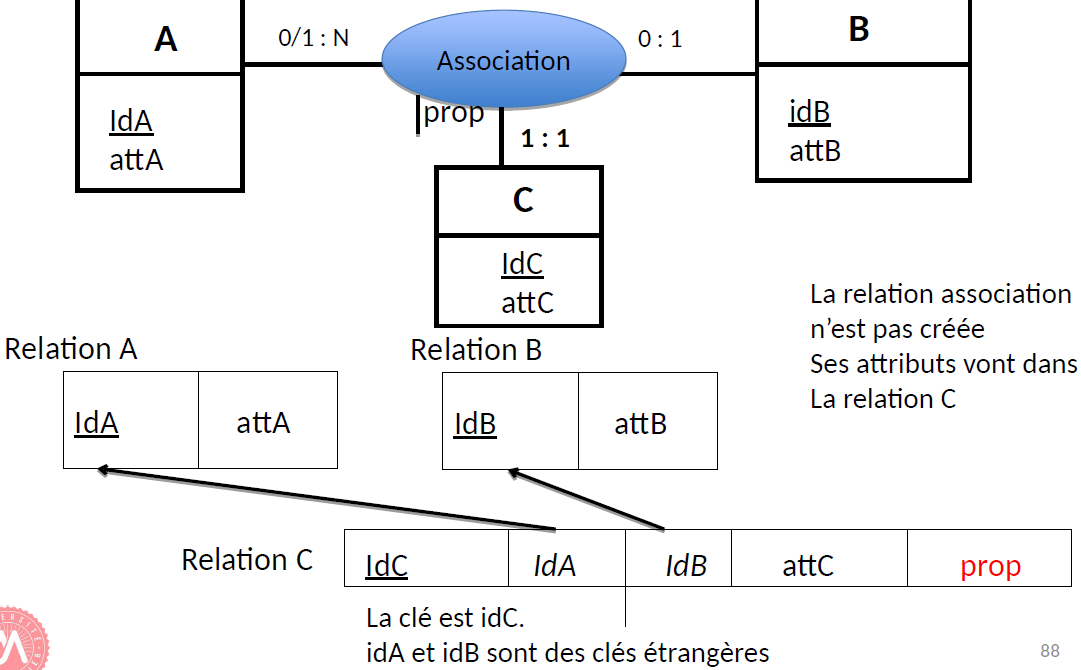
**Cas 1 cardinalité 1 : 1 sur une patte :** 

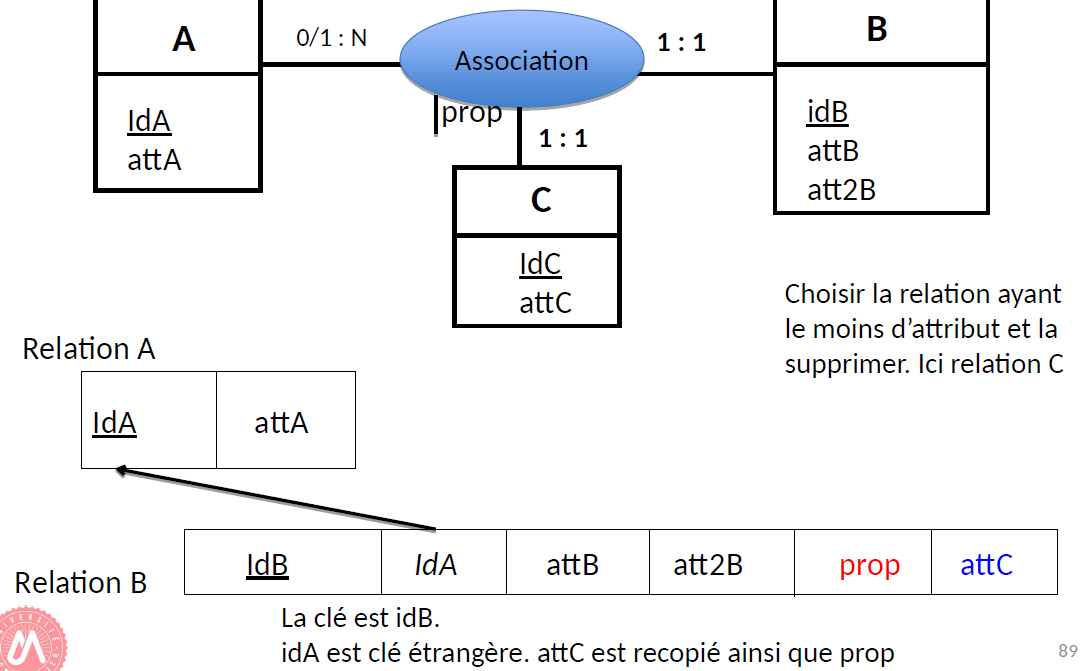
**Cas 2 cardinalité \_:N :** 

**Cas 3 : 0:1 sur une patte : **

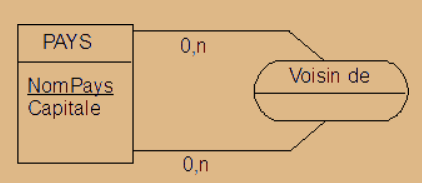
**Cas 1 : 1-1 et 0-1 : **

**Cas n-aires (:n, :n, :n) : **

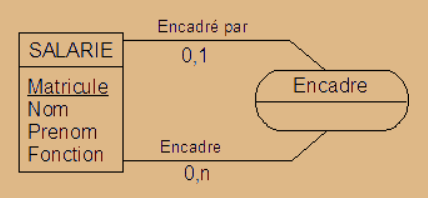
**Cas n-aires (:n, 0:1, 1:1) : **

**Cas n-aires (:n, 1:1, 1:1) : **

**Associations réflexives en relationnel :**

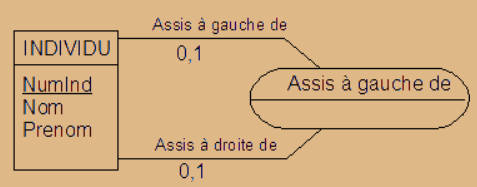
**cas 1 : n,n :  Voisin\_De (#NomPays,#VoisinDe)**

Une association réflexive [n,n] sur une entité E est traduite en une relation de même nom avec deux clés étrangères. L'une d'elles porte le nom de l'identifiant de l'entité et l'autre, le nom de l'association. La clef primaire de cette relation est constituée de ces deux attributs.

**cas : 1,n : **

**Salarié (Matricule,Nom,Prénom,Fonction,#Encadré\_Par)**

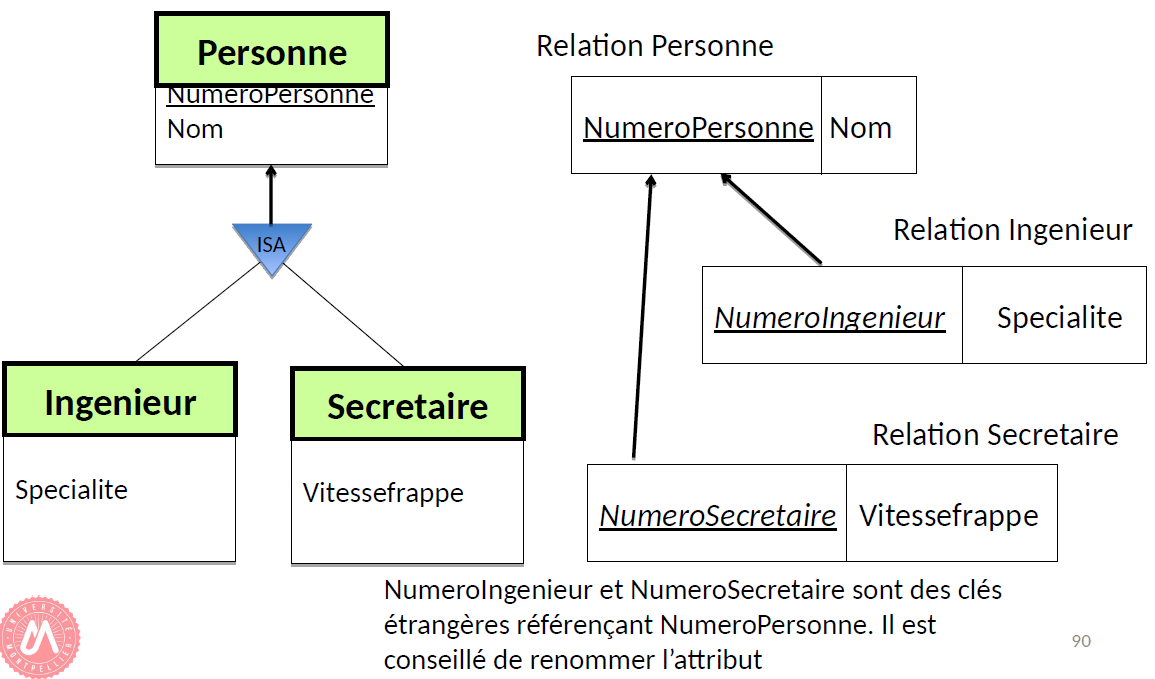
Une association réflexive [1,n] sur une entité est traduite en une clef étrangère dans la relation représentant cette entité. Le nom de cette clef étrangère est celui de l'association si c'est le nom de l'association qui étiquette le trait de cardinalité maximale 1, sinon c'est le nom de l'association réciproque.

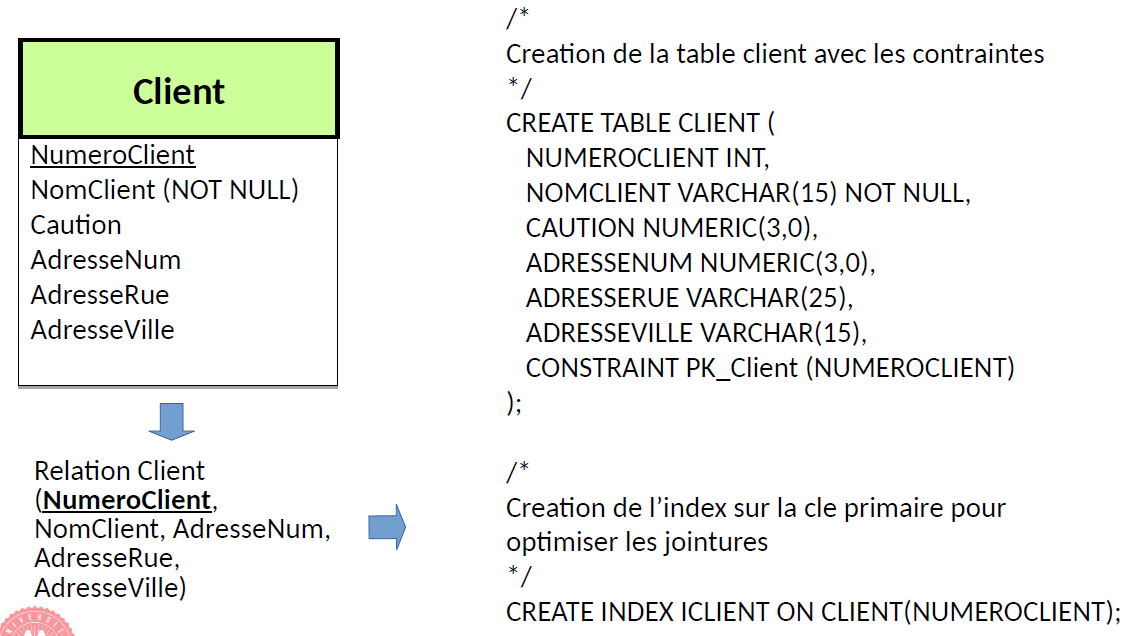
**cas : 1,1 :** 

**Individu (NumInd,Nom,Prénom,#Assis\_à\_Gauche\_De)**

Une association réflexive [1,1] sur une entité est traduite en une clef étrangère dans la relation représentant cette entité. Le nom de cette clef étrangère est celui de l'association.

**Les héritages :**





## Le langage SQL LMD :

<> = différent : SELECT DISTINCT business\_key FROM memory WHERE concept <> 'case' or a\_rib <> 'status' or value <> 'closed’

Null : WHERE att IS NULL ou bien WHERE att IS NOT NULL

**Condition d’existence :** WHERE EXISTS (REQUETE)

Est évaluée à vrai si la requête retourne un résultat. Attention pas de nom d’attribut – c’est une condition d’existence

SELECT \* FROM AVION WHERE EXISTS (SELECT \* FROM PILOTE WHERE Plnom=‘DUPONT’)

Est évalué à vrai s’il existe un pilote qui se nomme DUPONT dans la base

• Donner toutes les informations sur les Airbus dont le numéro est compris entre 100 et 150 et qui sont localisés à NICE, MARSEILLE, TOULOUSE ou BORDEAUX :

SELECT \* FROMAVION

WHERE Avnom LIKE 'Airbus%'

AND Avnum BETWEEN 100 AND 150

AND Loc IN ('NICE', 'MARSEILLE', 'TOULOUSE’, 'BORDEAUX') ;

• Attention le résultat d’une fonction d’agrégation retourne une seule valeur.

* Si la condition doit être vérifiée pour une des valeurs de la liste, on fait précéder la sous requête de IN ou =ANY
* Si la condition doit être vérifiée pour toutes les valeurs de la liste, on fait précéder la sous requête de t ALL où t est un opérateur de comparaison

Quels sont les avions du même nom et localisés au même endroit que l'avion Numéro 105 ?

SELECT \* FROM AVION WHERE (Avnom, Loc) = (SELECT Avnom, Loc FROM AVION WHERE Avnum = 105);

Jointure Inner Join :

SELECT Volnum, HD, HA FROM VOL INNER **JOIN** AVION **ON** VOL.Avnum = AVION.Avnum WHERE VD = 'PARIS' AND Avnom = ‘A320’ ;

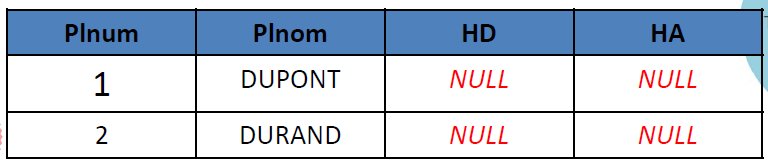
Alias dans le inner join :

SELECT \* FROM VOL JOIN PILOTE P ON VOL.Plnum = P.plnum WHERE P.ADR <> 'LYON';

Les jointures :

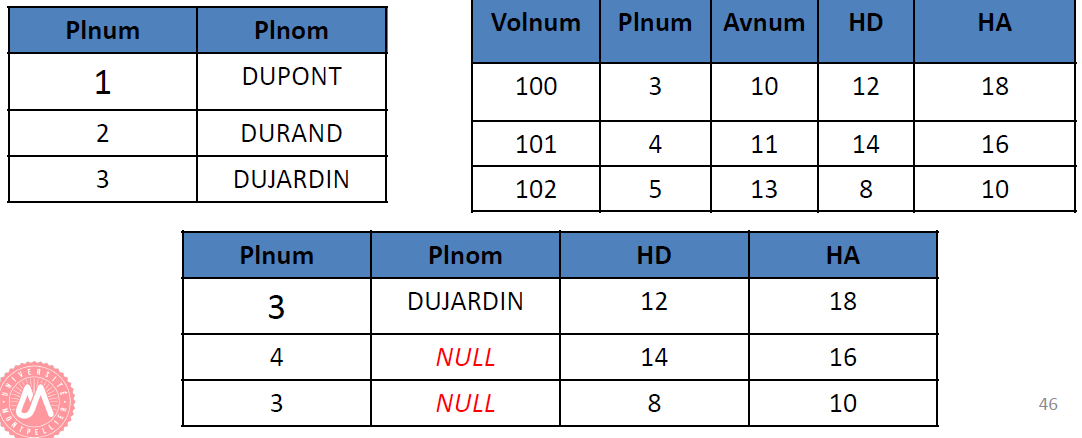
**• LEFT JOIN** : Intérêt permet de reporter tous les résultats de la relation de gauche même s’il n’y a pas de correspondance dans table de droite.

SELECT PILOTE.Plnum, Plnom, HD, HA FROM PILOTE LEFT JOIN VOL ON VOL.Plnum = PILOTE.Plnum WHERE HD IS NULL;



**• RIGHT JOIN** : reporter les résultats de droite même sans correspondance à gauche

SELECT PILOTE.Plnum, Plnom, HD, HA FROM PILOTE RIGHT JOIN VOL ON VOL.Plnum = PILOTE.Plnum;



**• FULL OUTER JOIN** : Intérêt permet de reporter tous les tous les résultats de la relation de droite et de gauche même s’il n’y a pas de correspondance Un NULL est attribué à droite ou à gauche.

SELECT PILOTE.Plnum, Volnum FROM PILOTE FULL JOIN VOL ON PILOTE.Plnum = VOL.Plnum;

Liste des pilotes qui habitent NICE et PARIS :

SELECT Plnom FROM PILOTE WHERE Adr='NICE'

**UNION**

SELECT Plnom FROM PILOTE WHERE Adr = 'PARIS';

Numéros des pilotes qui habitent à NICE et dont la ville de départ d’un vol est PARIS :

SELECT Plnum FROM PILOTE WHERE Adr='NICE'

**INTERSECT**

SELECT Plnum FROM PILOTE WHERE VD = 'PARIS' ;

Numéros des pilotes habitant Nice et n’assurant aucun vol au départ de Nice :

SELECT PlnumFROMPILOTE WHERE Adr = 'NICE'

**MINUS**

SELECT DISTINCT Plnum FROM VOL WHERE VD='NICE' ;

Mises à jours :

• Le pilote DUPONT change d’adresse et son salaire est augmenté de 10 % :

UPDATE PILOTE SET Adr='PARIS', Sal=Sal\*1.1 WHERE Plnom = 'DUPONT';

Insertion :

INSERT INTO nomrelation (list\_att) VALUES (list\_val) ;

Suppression :

DELETE FROM nomrelation WHERE condition;

DELETE FROM PILOTE WHERE Plnum=206;

**• Les attributs mentionnées dans le SELECT doivent être indiquées dans le GROUP BY**

**Requêtes corrélées :**

SELECT \* FROM PILOTE WHERE Sal =

(SELECT Sal FROM PILOTE WHERE Plnum=110)

• Quels sont les pilotes qui effectuent des vols ?

SELECT \* FROM PILOTE LesPilotes

WHERE NOT EXISTS (SELECT \* FROM Vol WHERE LesPilotes.Plnum=Vol.Plnum) ;

• Existe t‘il des homonymes parmi les pilotes ?

* Utilisation de la différence

SELECT p1.Plnum, p1.Plnom, p2.Plnum, p2.Plnom FROM PILOTE p1, PILOTE p2 WHERE p1.Plnom = p2.Plnom

MINUS

SELECT p1.Plnum, p1.Plnom, p2.Plnum, p2.Plnom FROM PILOTE p1, PILOTE p2 WHERE p1.Plnum = p2.Plnum;

* Une autre vision de la différence

SELECT p1.num, p1.nom FROM PILOTE p1, PILOTE p2 WHERE p1.nom = p2.nom

AND (p1.num, p1.nom, p2.num, p2.nom) NOT IN

(SELECT p3.num, p3.nom, p4.num, p4.nom FROM PILOTE p3, PILOTE p4 WHERE p3.num = p4.num);

* Avec une requête corrélée

SELECT \* FROM PILOTE p1 WHERE EXISTS (SELECT \* FROM PILOTE p2

WHERE p1.num != p2.num AND p1.nom = p2.nom);

* Avec un partitionnement

SELECT Plnum, Plnom FROM PILOTE WHERE Plnom IN (SELECT Plnom FROM PILOTE GROUP BY Plnom HAVING COUNT (\*) >1);

* Avec une condition sur les numéros – Résultats un peu différents

SELECT p1.Plnum, p2.Plnum, p1.Plnom FROM PILOTE p1, PILOTE P2 WHERE p1.Plnom=p2.Plnom AND p1.Plnum < p2.Plnum;

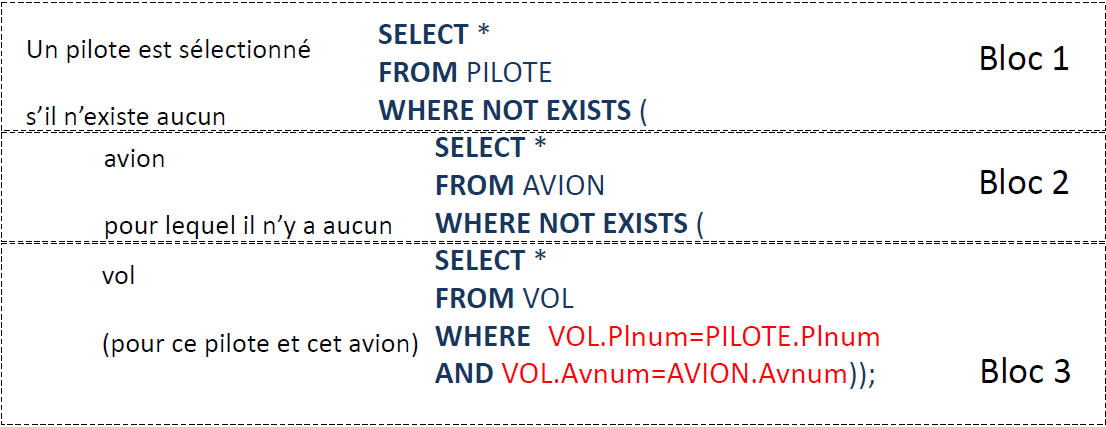
Divisions :

• Existe-t-il un pilote tel qu'il n'existe aucun avion de la compagnie qui ne soit pas conduit par ce pilote ?

SELECT \* FROM PILOTE WHERE NOT EXISTS (SELECT \* FROM AVION

WHERE NOT EXISTS (SELECT \*

FROM VOL WHERE VOL.Plnum=PILOTE.Plnum AND VOL.Avnum=AVION.Avnum));



• Les pilotes conduisant tous les airbus :

SELECT \* FROM PILOTE WHERE NOT EXISTS (SELECT \* FROM AVION

WHERE Avnom LIKE 'AIRBUS%' AND NOT EXISTS (SELECT \*

FROM VOL WHERE VOL. Plnum=PILOTE.Plnum AND VOL.Avnum=AVION.Avnum));

• Quels sont les pilotes qui conduisent autant d'avions que la compagnie en possède ?

SELECT Plnum FROM VOL GROUP BY Plnum HAVING COUNT (DISTINCT Avnum) = (SELECT COUNT(\*) FROM AVION);

Quels sont les x associés à tous les y de R ?

Quels sont les x tels qu’il n’existe pas de y qui ne soit pas associé à ce x ?

## Le langage SQL LDD :

• Le Langage de Définition des Données permet de gérer la définition d’une base de données et de tous les éléments qui la composent.

• Il permet par exemple de : Créer une relation, de modifier une relation ou de

supprimer une relation.

• Un schéma SQL (une base) est identifié par un nom de schéma

• CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] Nom\_Base [specification ...]

• Les termes DATABASE ou SCHÉMA peuvent être utilisés CREATE DATABASE PILOTE\_AVION\_VOL;

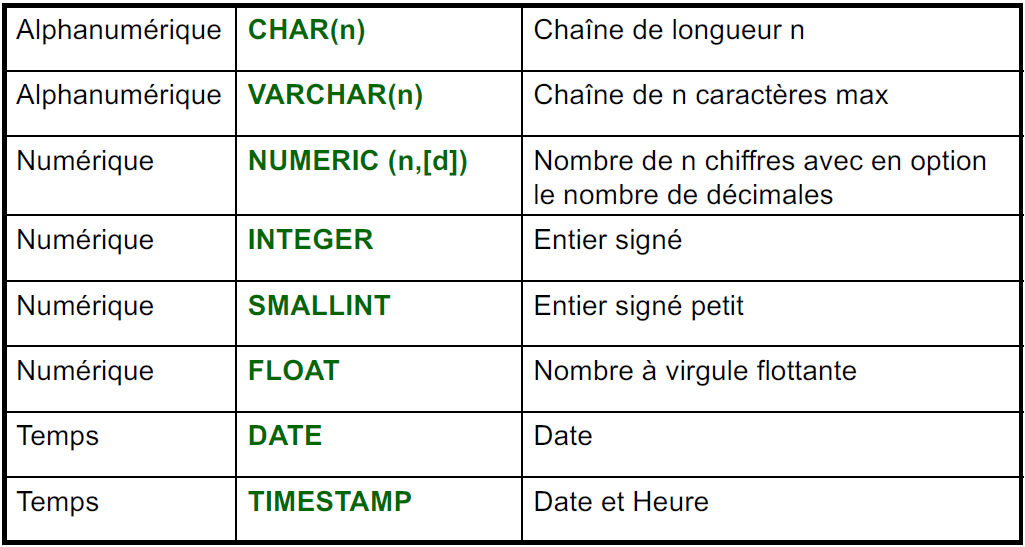
• CREATE TABLE a pour effet de créer une nouvelle relation, initialement vide, dans la base de données courante

• Il est possible de préciser un nom de schéma CREATE TABLE MONSCHÉMA.MATABLE;

• La table est alors créée dans le schéma « MONSCHEMA »

• La table créée appartient à l’utilisateur qui l’a créé

• Possibilité de spécifier des contraintes sur les valeurs d’une colonne (NOT NULL, UNIQUE, CHECK, …)



CREATE TABLE NOM\_DE\_LA\_TABLE (

<NOM COL1> <TYPE COL1> <CONTRAINTE COL1>,

<NOM COL2> <TYPE COL2> <CONTRAINTE COL2>,

…

<NOM COLn> <TYPE COLn> <CONTRAINTE COLn>

);

Insérer un tuples : SQL> INSERT INTO R VALUES ('14-02-2016');

• Il est possible de définir son propre domaine pour faciliter la lecture des schémas

CREATE DOMAIN Nom\_Domaine AS Type;

CREATE DOMAIN Adresse AS VARCHAR(20);

• **PRIMARY KEY** : pour spécifier une clé primaire

• **FOREIGN KEY** : pour spécifier une clé étrangère

• **REFERENCES** : pour les contraintes d’inclusion

• **CHECK** : contrainte générale

• **UNIQUE** : oblige le fait d’avoir une valeur unique

• **NOT NULL** : pour obliger à saisir une valeur

• **CONSTRAINT** : pour nommer une contrainte – Très utile

Il n’y a qu’une seule et unique clé primaire :

CREATE TABLE ECRIT(

Auteurnum NUMERIC(10),

ISBN NUMERIC(10),

Date DATE,

CONSTRAINT PK\_ECRIT

PRIMARY KEY (Auteurnum, ISBN)

);

• Il existe deux méthodes pour définir les clés étrangères :

* lors de la déclara\*on de l'attribut s’il est unique

att TYPE REFERENCES <relation>(aE’)

* avec la contrainte

FOREIGN KEY (att1, … att2) REFERENCES <relation> (att1’, …att2’)

• **NOT NULL** : impose de mettre une valeur pour l’attribut

• **UNIQUE** : impose une valeur différente de celles des autres attributs

Les contraintes check :

Elles permettent de prendre en compte la contrainte de domaine du modèle relationnel :

• CHECK (condition) où condition utilisent les opérateurs vus dans le select

• A BETWEEN a AND b

• A IN (a1, a2, ..an)

• A LIKE expression

• A >, <, =,… valeur

• DEFAULT valeur par défaut

**Création d’une relation :**

CREATE TABLE VOL (

Volnum NUMBER(7),

Plnum NUMBER(7),

…

CONSTRAINT PK\_VOL PRIMARY KEY (Volnum),

CONSTRAINT FK\_VOL\_PILOTE FOREIGN KEY (Plnum)

REFERENCES PILOTE(Plnum)

);

• **Convention** :

– Clé primaire : PK\_NOMRELATION

– Clé étrangère : FK\_NOMRELATIONDEPART\_NOMRELATIONARRIVEE

• Il est possible de créer une relation à partir d’une requête :

CREATE TABLE COPIEAVION AS (SELECT \* FROM AVION);

CREATE TABLE AVIONPARIS AS (SELECT \* FROM AVION WHERE Loc='PARIS');

La mise à jour des tuples :

• C’est une opération dangereuse ! Le SGBD doit garantir les contraintes de domaine, de clé primaire et de clé étrangère.

• Il existe une contrainte supplémentaire pour traiter ce problème :

**ON [DELETE|UPDATE] [CASCADE|SET NULL]**

* **DEFAULT** : rejet de la modification
* **CASCADE** : fait les mêmes changements que dans la

relation R. Si un tuple de R est effacé ou mis à jour alors tous les tuples de S qui référencent un tuple de R sont effacés ou mis à jour

* **SET NULL** : les tuples de S référençant un tuple de R qui est supprimé sont mis à NULL pour l’attribut clé de R

Les ON DELETE :

• **ON DELETE CASCADE**

* Objectif : supprimer automatiquement toutes les valeurs des attributs qui référencent la valeur.
* Conséquences : les tuples dans la relation associée sont supprimés.

• **ON DELETE SET NULL**

* Objectif : mettre à NULL toutes les valeurs de clés étrangères associées.
* Conséquences : possible si la clé étrangère associée est NULL. Impossible si la clé a la contrainte NOT NULL.

Les ON UPDATE :

• Cela veut dire que l’on change une clé primaire. Quelles en sont les conséquences ?

• **ON UPDATE CASCADE**

* Objectif : répercuter les modifications dans la relation associée
* Conséquences : les tuples associés sont modifiés

• **ON UPDATE SET NULL**

* Objectif : mettre à NULL toutes les valeurs associées
* Conséquences : NULL pour la clé étrangère associée

Exemple :

CREATE TABLE VOL(

Volnum NUMBER(7),

Plnum NUMBER(7),

CONSTRAINT PK\_VOL PRIMARY KEY(Volnum),

CONSTRAINT FK\_PILOTE\_VOL FOREIGN KEY(Plnum)

REFERENCES PILOTE(Plnum) ON UPDATE CASCADE

);

UPDATE PILOTE SET Plnum = 20 WHERE Plnum = 10;

Modifiera automatiquement dans VOL les tuples où le numéro de pilote était égal à 10.

Les indexs :

• Les **index** sont utilisés pour accélérer les accès à une relation

• Ils sont là pour optimiser les requêtes :

CREATE INDEX nom\_index ON nom\_table (nomdesattributs);

CREATE INDEX IPILOTE ON PILOTE (PLNUM);

• La plupart du temps lorsque vous créez une contrainte de clé primaire, étrangère ou une

contrainte d'unicité, le SGBDR implante automatiquement un index pour assurer la mécanisme de contrainte avec des performances correctes.

• Pour une relation donnée, il convient d'i**ndexer dans l'ordre** :

* les colonnes composant la clé primaire
* les colonnes composant les clés étrangères
* les colonnes composant les contraintes d'unicité
* les colonnes dotées de contraintes de validité
* les colonnes fréquemment mises en relation, indépendamment des jointures
* les colonnes les plus sollicitées par les recherches

• Lors de la création d’une relation il faut a minima :

* Créer un index sur la clé primaire
* Créer un index sur les clés étrangères

Suppression d’une base :

**DROP SCHEMA nom\_base [CASCADE|RESTRICT];**

• CASCADE : effacer toute la base

• RESTRICT : effacer seulement si la base n’a plus de tuples

Suppression d’une relation :

• DROP TABLE nom de la relation [CASCADE|RESTRICT];

DROP TABLE AVION;

Modification d’une relation :

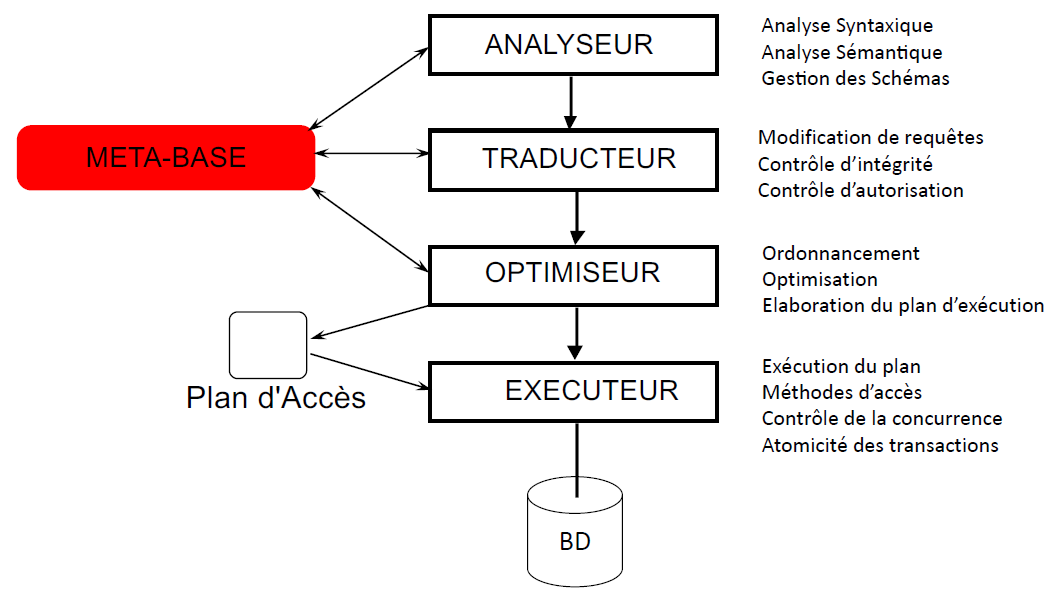
• ALTER TABLE nom\_colonne [ADD|MODIFY|DROP|RENAME][CASCADE|RESTRICT];

* ALTER TABLE PILOTE ADD TEL VARCHAR(10);
* ALTER TABLE PILOTE MODIFY (Loc DEFAULT 'PARIS');

• Le cas de **RENAME** (attribut, relation)

* ALTER TABLE PILOTE RENAME COLUMN NOM TO LENOM;
* ALTER TABLE PILOTE RENAME TO PILOTES;

Structure d’un SGBD :



• Attention dépendant du SGBD :

* Mysql : SHOW TABLES;
* Sybase : SELECT \* FROM SYSTABLES;
* Oracle : SELECT \* FROM ALL\_TABLES;

• Lister les tables du schéma **de l'utilisateur courant :**

* SELECT TABLE\_NAME FROM USER\_TABLES;

• Lister les tables **accessibles par l'utilisateur courant** :

* SELECT TABLE\_NAME FROM ALL\_TABLES;

• Lister les tables **d’un utilisateur DUPONT** :

* SELECT TABLE\_ NAME FROM ALL\_TABLES WHERE OWNER='DUPONT';

• Lister **toutes les tables** (il faut être ADMINISTRATEUR) :

* SELECT TABLE\_NAME FROM DBA\_TABLES;

• Il est possible d’interroger la méta-base pour connaître l’ensemble des contraintes de type **clé primaire** :

SELECT \*FROM ALL\_CONSTRAINTS WHERE CONSTRAINT\_NAME LIKE 'PK\_%';

Architecture ANSI/SPARC :

• **Externes** (vues)

* description des entités et associations ou plutôt des relations vues par un utilisateur

• **Conceptuel**

* description des entités et associations du monde réel

• **Interne**

* implémentation physique des entités et associations dans la base sous la forme de relations et d’index

La notion de vues :

**• Rôle de sécurité**

* L’utilisateur ne peut accéder qu’aux données des vues auxquelles il a le droit d’accès

**• Contraintes d’intégrité**

* Mise à jour au travers de vues

Vue : Base de données virtuelle dont le schéma et le contenu sont dérivés de la base réelle par un ensemble de requêtes. Une vue est donc un ensemble de relations déduites d'une base de données, par composition des relations de la base.

**Création d’une vue :**

CREATE VIEW <NOM DE VUE> [ (LISTE D'ATTRIBUT)] AS <REQUETE> [WITH CHECK OPTION]

* La clause WITH CHECK OPTION permet de spécifier que les tuples de la vue insérés ou mis à jour doivent satisfaire aux conditions de la requête.

**Destruction d’une vue :**

– DROP VIEW <NOM DE VUE>

Une vue n’a pas d'existence physique. Une destruction n'entraîne pas la suppression de tuples de la base.

Exemple création d’une vue pour les pilotes niçois :

CREATE VIEW PILOTESNICOIS (Plnum, Plnom, Sal) AS SELECT Plnum, Plnom, Sal FROM PILOTE WHERE Adr = 'NICE';

**Mise à jour des vues :**

• En pratique : seuls les attributs d’une table de la base doivent apparaître dans la vue

• Imposer que la clé de la table soit présente

• Tenir compte des problèmes potentiels d’insertion, de modification ou de suppression

• Importance de la clause WITH CHECK OPTION : forcer à faire la vérification

## Dépendances fonctionnelles et normalisation :

Il existe de nombreuse méthode pour concevoir une base de données :

* Empirique (Danger)
* Semi Empirique (EA)
* Formelle (Normalisation relationnel)

Objectif principal : Trouver un bon schéma relationnel. La qualité d’un schéma se mesure lors des opérations de mise à jour.

Les anomalies de modification :

* Modification du salaire des pilotes : autant de modifications qu’il ya de pilotes

Les anomalies d’insertion :

* Pour stocker le salaire des contrôleurs il faut qu’il y ait au moins une catégorie d’employés dans cette catégorie (pas de clé primaire nulle)

Les anomalies de suppression :

* Si suppression de DURANT on perd l’information sur le salaire des mécaniciens.

Pour éliminer ces anomalies et avoir un bon schéma relationnel, il faut normaliser la relation en la décomposant en plusieurs relations. S’il y a redondance, comment décomposer la relation ? Y’a t’il de l’information perdue lors de la décomposition.

Il existe une **dépendance fonctionnelle** (DF) entre X et Y, notée X→Y si et seulement si

* ∀ t1,t2∈ R
* si t1(X) = t2(X) alors t1(Y) = t2(Y)

Les DF sont des propriétés sémantiques donc s’appliquent à toute extension de R.

Par exemple, le salaire dépend de la catégorie. Donc si t1(Categorie) = t2(Categorie) alors t1(Salaire) = t2(Salaire).

Dans une relation, tout attribut est en DF avec la clé primaire.

Num->Nom (à tout numéro correspond un nom)

Num->Catégorie (à tout numéro correspond une catégorie)

* Catégorie **est déterminé par** Num
* Num **détermine** Nom, Catégorie

**Axiomes d’Armstrong :**

**• P1 : Réflexivité – si YcX alors X → Y (donc X → X) DF triviale**

Si A,B → A,B alors A,B → A et A,B → B

**• P2 : Augmentation – Si X → Y alors X,Z → Y,Z où Z c U**

Si A,B → C alors A,B,C → C

Si A,B → D alors A,B,C → CD

**• P3 : Transitivité – Si X → Y et Y → Z alors X → Z**

Si A,B → C et C → D alors A,B → D

**Règles d’inférences déduites des axiomes d’Armstrong**

• P4 : **Pseudo-transitivité** – Si X → Y et Y, Z → W alors X,Z → W

• P5 : **Union** – Si X → Y et X → Z alors X → Y,Z

• P6 : **Décomposition** – Si X → Y,Z alors X → Y et X → Z

Soit R (A, B, C, D, E, G, H) et

F= {A,B → C ; B → D ; C,D → E ; G → A ; D → H} :

**B → H**

– car B → D et D → H (P3)

**B,G → C**

– car G → A et par augmentation (P2) on a B,G → A,B. De plus A,B → C donc par transitivité (P3) on a B,G → C

**A,B → E**

– car B → D et par augmentation (P2) on a A,B → A,D donc par décomposition (P6) on a A,B → D or nous avons A,B → C par union (P5) on a A,B → CD et nous savons que

C,D → E donc par transiHvité́(P3) on obtient A,B → E

**Fermeture d’un ensemble de DF :**

Soit F un ensembles de DF et X → Y une DF. F implique X → Y, noté : F ⊨ X → Y, signifie que toute instance de relation sur R qui satisfait les dépendances dans F satisfait aussi X →Y.

**Fermeture transitive :**

La fermeture transitive (ou clôture) d'un ensemble de dépendances fonctionnelles, F, est ce même ensemble enrichi (F+) de toutes les dépendances fonctionnelles que l’on peut dériver en appliquant les axiomes d’Armstrong. : F+ contient toutes les DF impliquées par F : F+ = {X → Y |F ⊨ X → Y}. Comme la fermeture contient trop d’ensemble d’attributs, on calcule des fermetures d’ensembles d’attributs.

La fermeture transitive d’un ensemble d’attributs X est notée X+ elle contient tous les attributs qui dépendent des attributs dans X [X]+F ={A| F|= X→A}.

F = {A →B ; B →C}, R(A,B,C) [A]+F = { A, B, C }

A partit de A on peut déterminer l’ensemble des attributs : A →C est dans la fermeture de F

• A →C est dans la fermeture de F

• A est une **surclé** (et clé) de R

**Algorithme de calcul d’une fermeture transitive :**

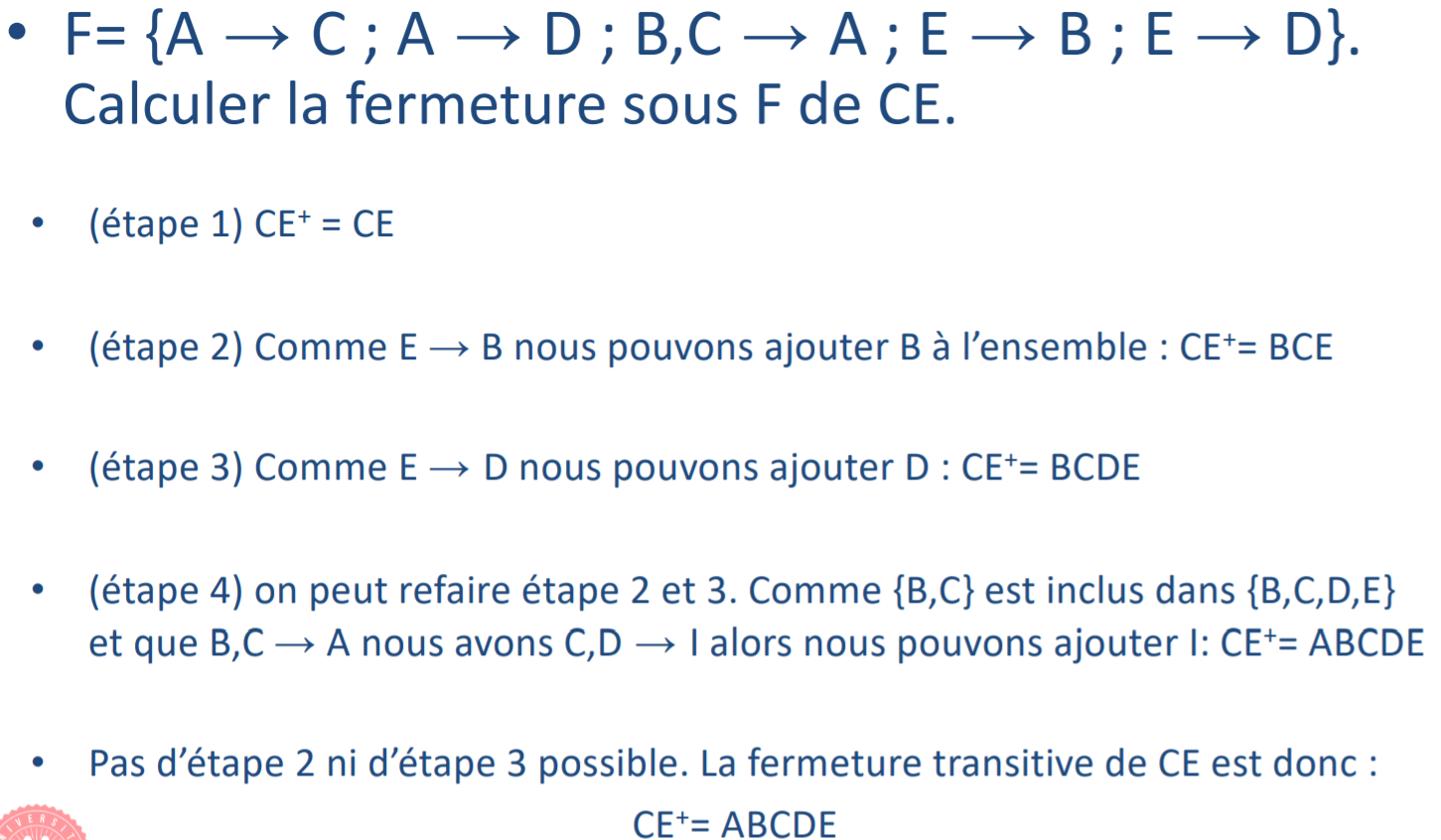
1. Initialiser X+ à X

2. Trouver une DF (G → D) ∈ F possédant en partie gauche des attributs inclus dans X+

3. Ajouter dans X+ les attributs situés en partie droite de la DF

4. Répéter 2) et 3) jusqu'à ce que X+ ne puisse plus évoluer

Exemple :



**Equivalence :**

Deux ensembles de DF différents peuvent exprimer les même contraintes :

F = { A,B → D ; D →C ; C →D ; A,B →C }

G ={A,B →C ; D →C ; C →D}

* F et G sont équivalents (expriment les mêmes contraintes). On note : F ≡ G
* G est plus « compacte »
* F est équivalent à G(F ≡ G) ssi F+ = G+

**Dépendance fonctionnelle élémentaire** : La seconde condition indique que X est « la plus

petite quantité d'informations donnant Y ». Il n’y a pas d’attribut inutile dans la partie gauche

**Dépendance fonctionnelle directe** : Soit R(U) une relation, soit X et Y Ì U, tels

que : X → Y. La dépendance X → Y est directe s'il n'existe pas Z dans R distinct de X et Y tel que X → Z et Z → Y. La dépendance n'est pas obtenue par transitivité

**Dépendance fonctionnelle triviale et simple** : La dépendance X → Y est triviale si Y – X est vide. Une dépendance fonctionnelle est simple si elle ne comporte qu'un seul attribut en partie droite et si elle n'est pas triviale. X → A1,A2,...,An ⇔{X → A1 ; X → A2 ;...;X → An }.

**Couverture minimale :** sous ensemble minimum de dépendances fonctionnelles élémentaires qui permettent de générer toutes les autres.Tout ensemble de dépendances fonctionnelles possède une couverture minimale (pas forcément unique).

**Algorithme pour trouver une couverture minimale :**

1. Décomposer chaque DF pour avoir un seul attribut à droite (P6). (les cibles de DF n’ont qu’un attribut - DF simples)

2. Supprimer les attributs en surnombre à gauche : Pour tout X → Y, s’il existe un Z⊆X tel que Z → Y alors remplacer X → Y par Z → Y (propriétés P1, P3 et P4 ) (pas d’attribut inutile dans les DF de M – DF élémentaires)

3. Supprimer les DF redondantes (qu’on peut obtenir par les axiomes d’Armstrong – DF directes)

Soit F un ensemble de DF, soit f appartient à F, la DF A, B, C, D … → Y A est un **attribut inutile** dans f si on peut engendrer B, C, D, … → Y à partir des DF de F et des propriétés P1, P3 et P4 (réflexivité, transitivité, pseudo transitivité).

Une clé d'une relation R(A1, ..., An) est un sous ensemble X des attributs de la relation R tel que les deux conditions ci- dessous sont vérifiées :

1. X → A1, ..., An

2. Il n'existe pas de Y ⊂ X tel que Y → A1, ..., An

Un attribut clé est un attribut qui appartient à ce4e clé et un attribut non clé est un attribut qui n’y appartient pas

**Super clé** : un ensemble d'attributs satisfaisant la 1er propriété constitue une super clé de R

– Une super clé de R contient donc une clé de R

– Une clé de R est une super clé minimale de R

– Si X=U, la relation est dite « toute clé » : la clé est composée de l’ensemble des attributs

**Algorithme de recherche de clés :**

1. Recherche de la couverture minimale M

2. Regroupement des DF de M

– Réunir dans un même ensemble Ei toutes les DF ayant même source (autant de Ei que de source de DF différentes)

3. Regroupement des Ei

– On regroupe dans un même ensemble les DF de Ei et Ej s’ils contiennent des DF réciproques (X → Y et Y → X)

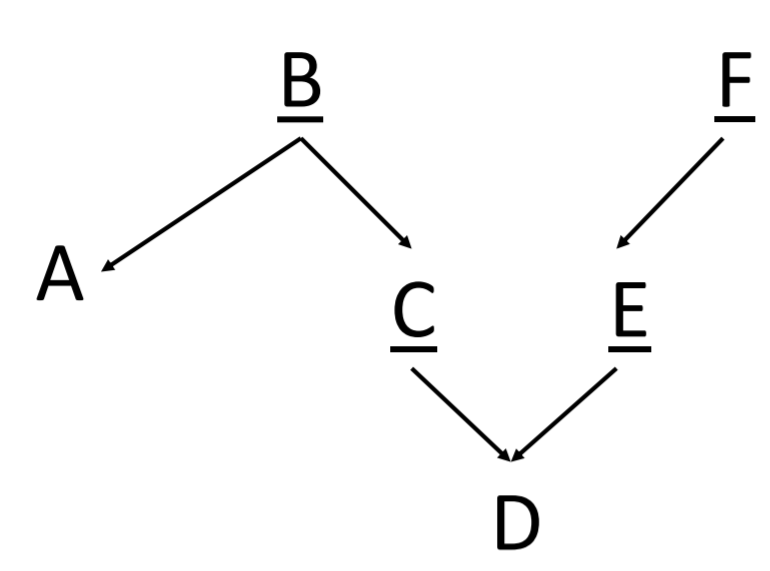
4. Création des relations – Réunir dans un même ensemble toutes les DF ayant même

source (autant de Ei que de source de DF différentes)

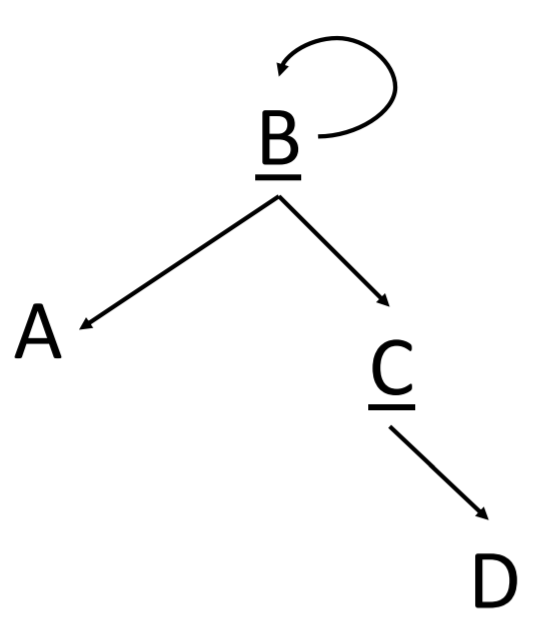
**Graphes des dépendances fonctionnelles :**

* Le ou les attributs clés primaires sont soulignés
* Il y a une flèche du ou des attributs clés primaires vers les attributs non clés primaires

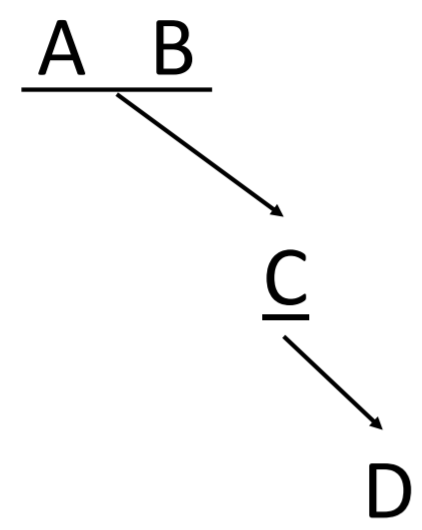
R1 (B,C,A) ; R2 (C, D) ; R3 (E, D) ; R4 (F, E)



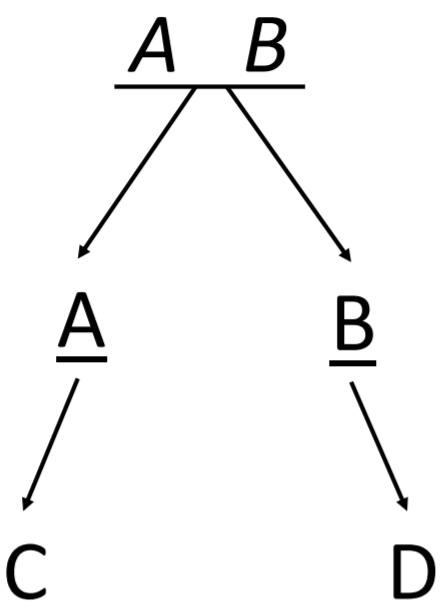
Auto Jointure R1 (B,C,A, refB) ; R2 (C, D)



Clé primaire multi-attributs R1 (A,B,C) ; R2 (C, D)



Clé primaire multi-attributs avec relations complexes R1 (A,B) ; R2 (A, C) ; R3 (B, D)



L’application des axiomes et le calcul de la fermeture est la même chose en fait.

**Attributs inutiles :**

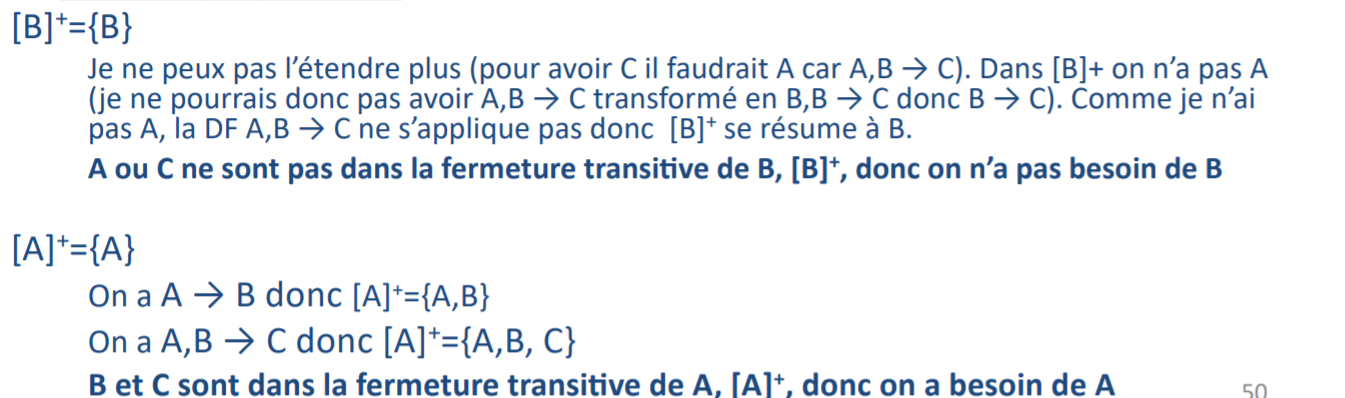
Intuition : A → B ; A,B → C

* Si B est inutile cela revient à dire que sans lui je peux déterminer l’attribut C donc A → B ; A,B → C
* Si A est inutile cela revient à dire que sans lui je peux déterminer l’attribut C donc A → B ; A,B → C

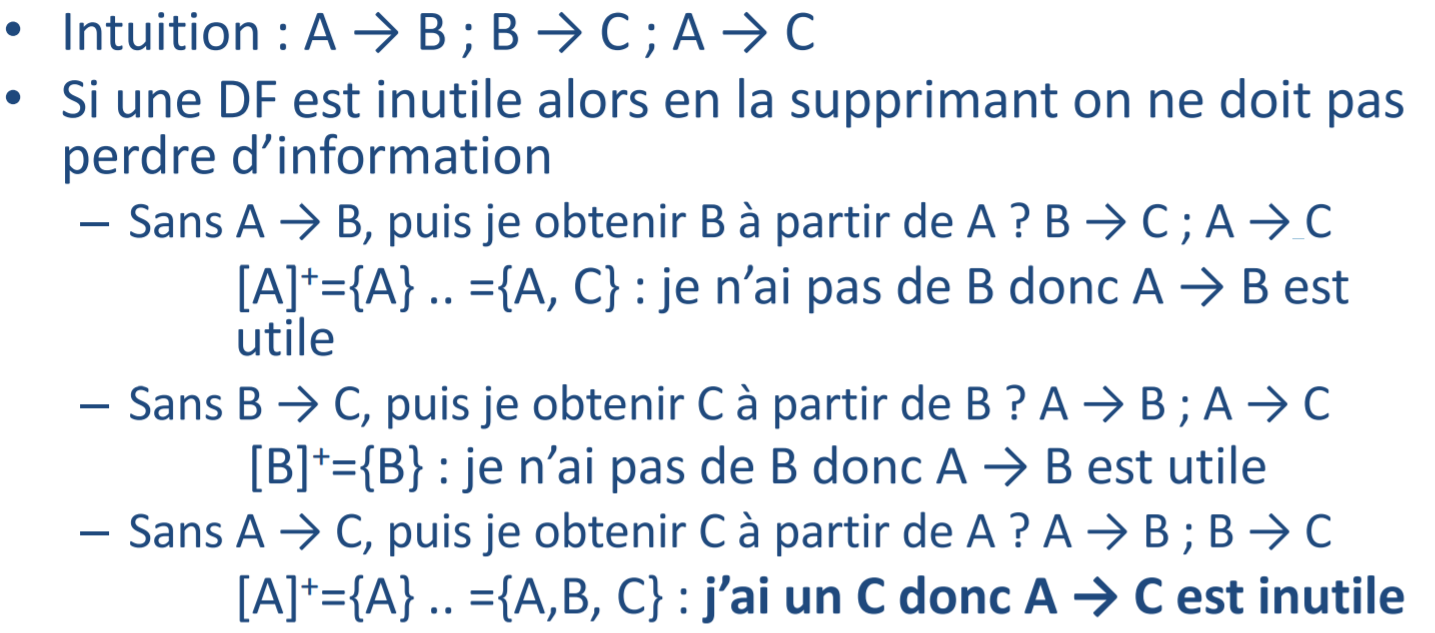
On regarde : A,B → C – On calcule [B]+ et [A]+

• Si A ou C (la cible de la DF) est dans [B]+ alors cela veut dire qu’on peut atteindre A ou C directement à partir de B. On ne peut pas l’enlever.

• Si B ou C (la cible de la DF) est dans [A]+ alors cela veut dire qu’on peut atteindre B ou C directement à partir de A. On ne peut pas l’enlever.



Dépendances inutiles :



**Décomposition :**

Critères attendus de la décomposition :

* Décomposition sans perte d'information
* Décomposition préservant les DF

Toute relation R(X,Y,Z) est décomposable sans perte d’information en R1(X,Y) et R2(X,Z) s’il existe une DF telle que X → Y

**Exemple :**

Soit la relation **Entreprise (Ville, Rue, Code) et F = Ville, Rue → Code ; Code -> Ville**

La décomposition de Entreprise en **R1(Ville, Code)** et **R2(Rue, Code)** évite la redondance Ville, Code et est une décomposition qui est sans perte d'information mais qui elle ne préserve pas la dépendance fonctionnelle : **Ville, Rue → Code**

**Première forme normale :**

Une relation est en 1FN ssi tous ses attributs sont atomiques (mono-valués). Créer autant d’attributs que le nombre maximum de valeurs (stockage horizontal) Personne (Num, Nom, Prenom1, Prenom2, Prenom3). Créer une nouvelle relation comprenant la CP de la relation initiale et l’attribut multi-valué. Attention : éliminer l’attribut de la relation initiale.

**Seconde forme normale :**

Une relation est en seconde forme normale (2FN) ssi

* Elle est en 1FN
* Tout attribut n’appartenant pas à la clé primaire est en DF totale avec la clé

Isoler la DF responsable dans une nouvelle relation. Elle devient CP dans la relation initiale

• Éliminer l’attribut cible de la DF dans la relation initiale.

• Attention : la seconde forme normale implique que :

« Tout attribut n’appartenant pas à la clé primaire est en DF totale avec la clé ». Ceci doit être vrai pour les clés candidates.

**Troisième Forme Normale :**

Une relation est en 3FN ssi

* Elle est en 2FN
* Elle ne contient pas de DF transitive entre attributs non clés

Isoler la DF transitive dans une nouvelle relation

• Eliminer l’attribut cible de la DF dans la nouvelle relation

**Toute relation R** admet au moins une décomposition en 3FN telle que:

* La décomposition préserve les DF
* Toutes les composantes sont en 3FN

• **Conséquences** : Il est souhaitable que les relations soient en 3FN car il existe toujours une décomposition sans perte d'information et préservant les DF d'un schéma en 3FN

Une relation est en **BCFN** (Boyce and Codd Normal Form) ssi elle est en 3FN et qu’aucun attribut de la clé ne dépend d’un attribut non clé. R (A1, A2, A3) est en BCFN s’il n’existe pas A3 → A1. Toute relation admet une décomposition en BCFN sans perte d’information Une décomposition BCFN ne conserve pas les DF.

Souvent au niveau 3NF et BCNF :

* Plus d’anomalie de stockage
* Modification : modification du salaire des pilotes pour tous
* Insertion : on peut stocker le salaire d’un contrôleur sans avoir un employé de cette catégorie
* Suppression : si DURANT est supprimé on conserve l’information sur le salaire des mécaniciens

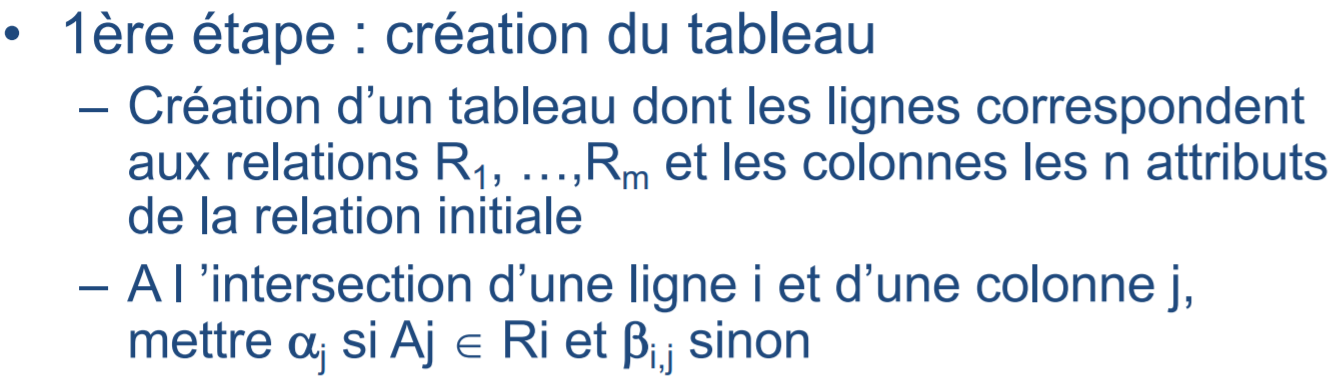
**Dépendances multi-valuées :** A chaque valeur de X correspond toujours le même

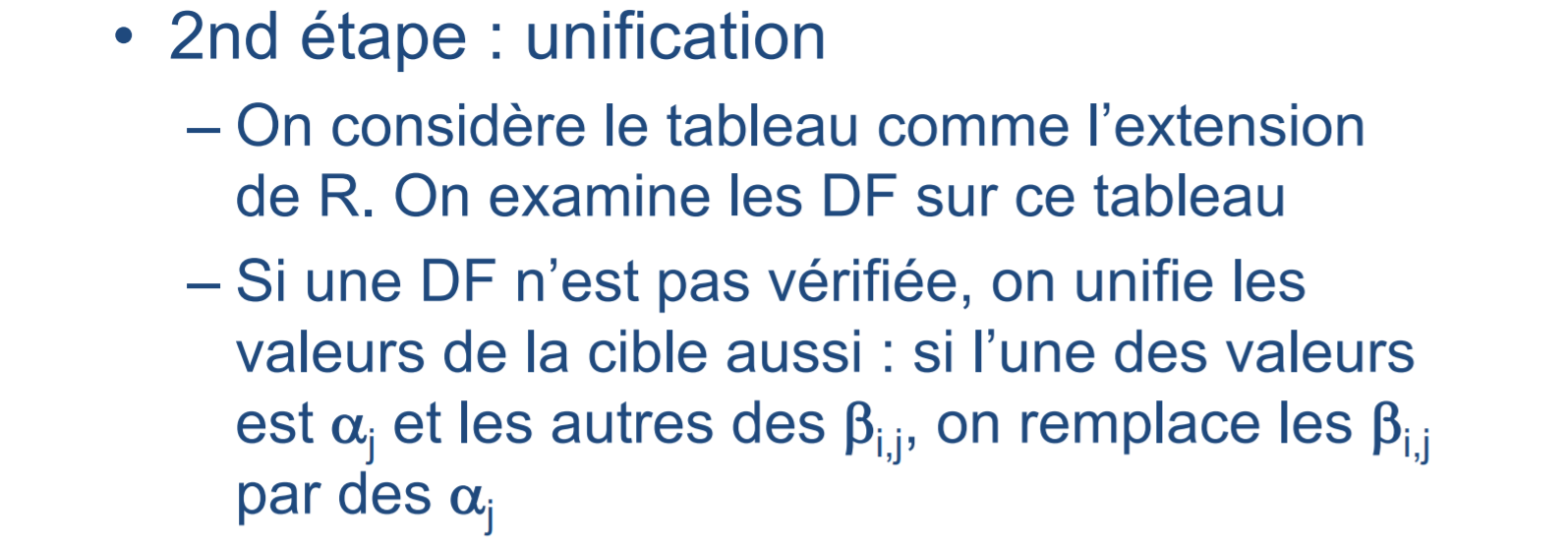
ensemble de valeurs de Y et cet ensemble de valeurs ne dépend pas de Z. “Un étudiant peut faire plusieurs sports et parler plusieurs langues”, NumEtudiant ↠ Sport, Langue.

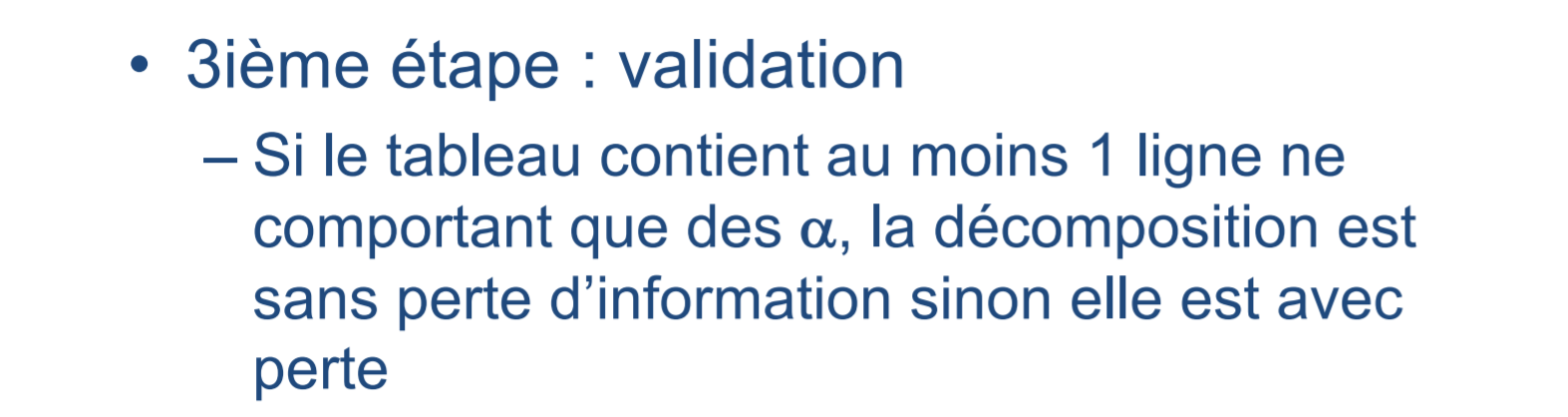
Les dépendances multi-valuées sont une généralisation des dépendances fonctionnelles :

si X → Y alors X↠Y

**Vérification qu’une décomposition est sans perte d’information :**







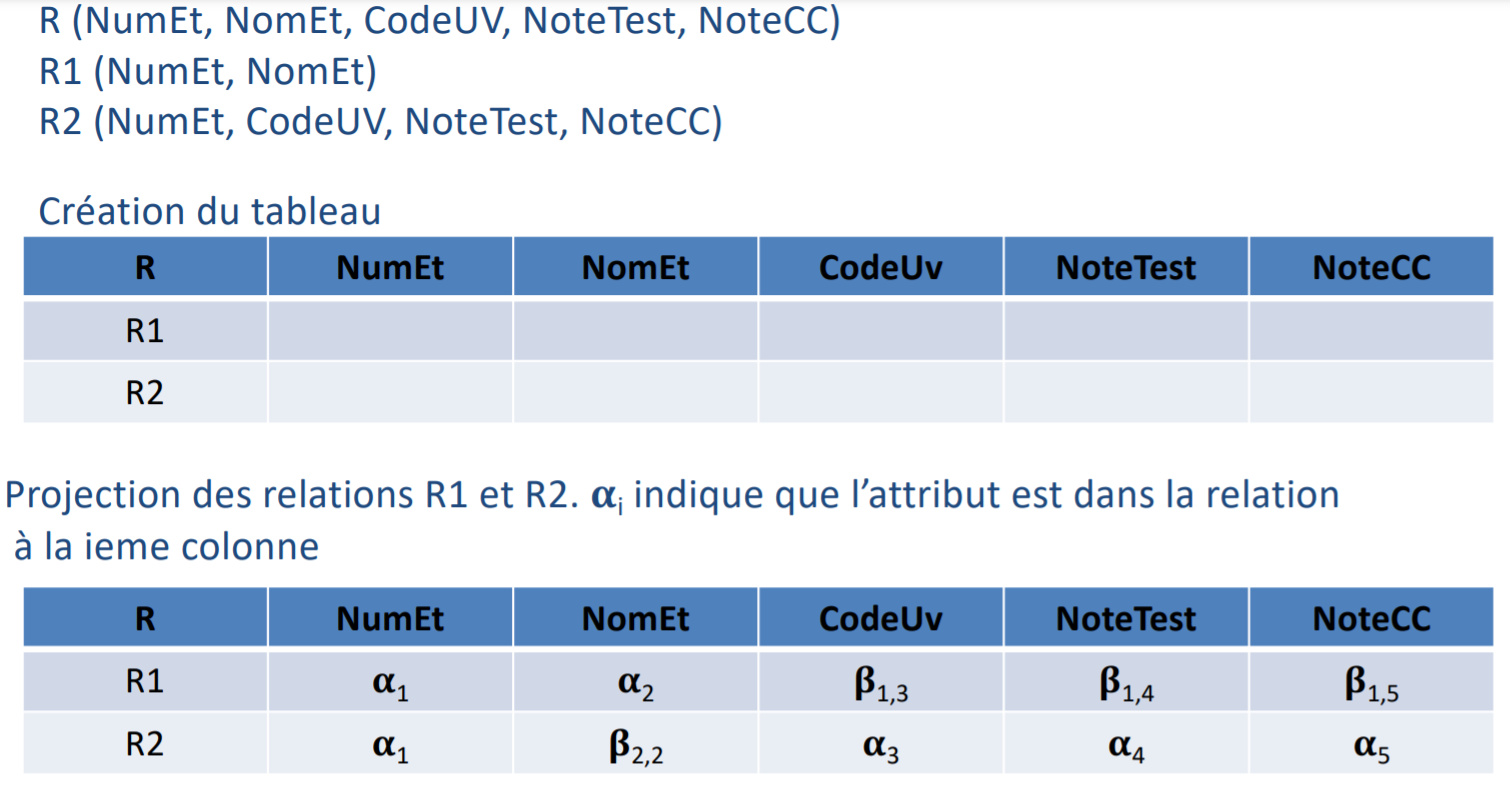
R (NumEt, NomEt, CodeUV, NoteTest, NoteCC) avec

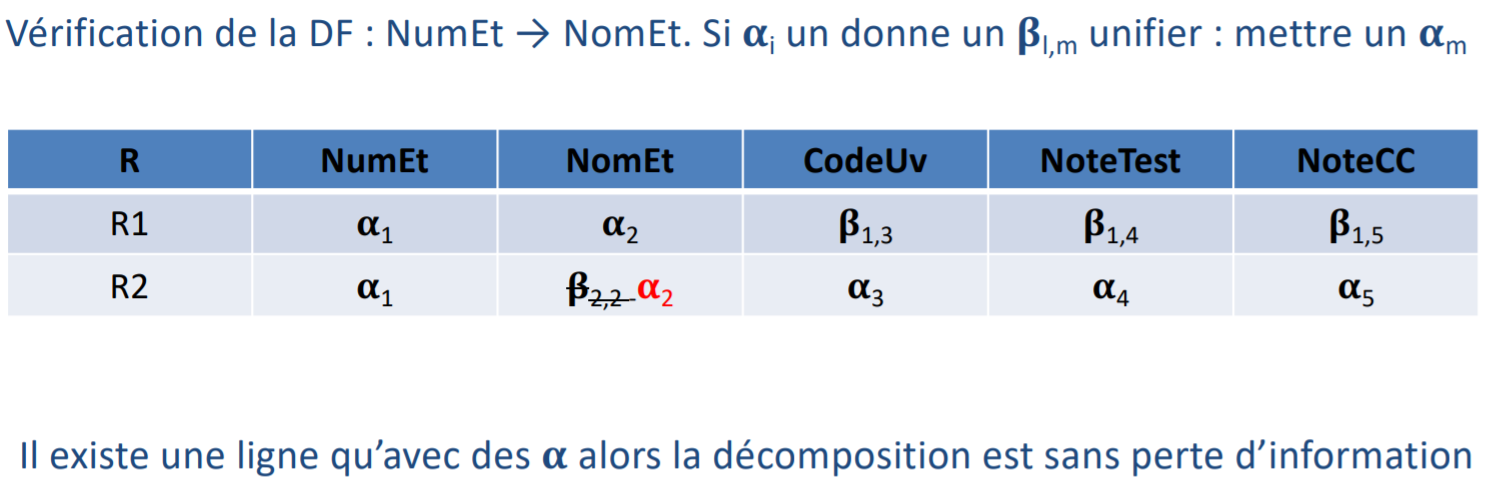
* NumEt → NomEt
* NumEt, CodeUV → NoteTest, NoteCC

• Décomposé en

* R1 (NumEt, NomEt)
* R2 (NumEt, CodeUV, NoteTest, NoteCC)

Donc on créer le tableau suivant.





## PL/SQL :

Programming language with SQL, langage de programmation procédurale. Propriétaire Oracle. Permet de passer d’un monde ensembliste à un monde enregistrement par enregistrement.

**Côté serveur**

* Offre la possibilité de définir des objets persistants : procédures, fonctions, triggers

**Côté client**

* Permet d’écrire des blocs PL/SQL anonymes
* Utilisable pour le développement d’interfaces graphiques, de masques de saisie (SQLForms), etc

La casse n’est pas importante. Pour les commentaires ce sont des -- pour les lignes et /\*.

**Un bloc PL/SQL :**

[DECLARE

Liste déclarations de variables, constantes, curseurs, exceptions]

[BEGIN]

Liste des instructions – Corps du bloc PL/SQL

[EXCEPTION

Gestion des exceptions]

[END] ;

/ -> le / indique exécution du bloc PL/SQL

*Les variables peuvent être de types suivants :*

* Scalaire, recevant une valeur de type SQL (CHAR, NUMBER, VARCHAR, …) ou de type PL/SQL (sous type prédéfini : INTEGER ou défini par l’utilisateur)
* Composé (RECORD, collection, types objets)
* Référence (REF) ou LOB (pour les données de grandes tailles)

Les contraintes **NOT NULL** doivent être suivies d’une clause d’initialisation

**Les déclarations multiples** ne sont pas autorisées : nom, prenom VARCHAR2(10); - Interdit.

Les affectations des variables dans le bloc BEGIN … END suivent la syntaxe classique := un\_nombre:=6;

**Record** : semblable à une structure C. Tous les types sont de type SQL. Une variable de type record peut ressembler à une ligne dans une relation :

* Type collections : TABLE, VARRAY (relationnel-objet)

Sont déclarés par : **attribut%TYPE** • Signifie « du même type que » :

numero PILOTE.Plnum%TYPE ; -- numero est du même type que l’attribut Plnum de la relation PILOTE

Sont déclarés par : **attribut%ROWTYPE** • Signifie « du même type d’enregistrement que » :

un\_pilote PILOTE%ROWTYPE ;

-- un\_pilote est du même type que la relation pilote

-- c’est à dire (Plnum, Plnom, Adr, Sal)

-- on peut accéder à ses attributs par un\_pilote.Plnum

**Notion de sous type :**

Chaque type prédéfini possède ses caractéristiques (domaine, opérateurs) • Un sous type permet de restreindre les caractéristiques • Il en existe des prédéfinis : INTEGER, CHARACTER, POSITIVE • Il est possible de créer ses propres sous types :

SUBTYPE nom\_sous\_type IS type\_de\_base [(contraintes)] [NOT NULL];

Les opérateurs de SQL sont aussi valides en PL/SQL.

**Instructions conditionnelles :**

IF <condition> THEN [BEGIN] <instructions> [END]

[ELSIF <condition> THEN [BEGIN] <instructions> [END] ]

[ELSE [BEGIN] <instructions> [END]]

END IF;

**Les itérations :**

Boucle **FOR** :

**FOR** <compteur> IN <borne\_inf> .. <borne\_sup>

**LOOP**

<liste\_instructions>

**END LOOP ;**

/\* il est inutile de déclarer <compteur> \*/

Boucle **WHILE** :

**WHILE**

<condition>

**LOOP**

<liste\_instructions>

**END LOOP ;**

Il existe aussi la possibilité de sortir avec un EXIT WHEN :

Boucle LOOP :

**LOOP**

<liste\_instructions>

**EXIT WHEN** <condition>

**END LOOP ;**

Exemple : Affiche les 10 premiers nombres à l’écran.

| BEGIN FOR num IN 0..10 LOOP  DBMS\_OUTPUT.put\_line(TO\_CHAR(num)) ; END LOOP ; END ; / |
| --- |

Exemple : insère dans la relation RESULTAT les 10 premières valeurs de 0 à 9

| DECLARE  NUM NUMBER(2) := 0 BEGIN LOOP  INSERT INTO RESULTAT VALUES (NUM)  NUM := NUM+1 ; EXIT WHEN NUM > 10 ; END LOOP ; END ; / |
| --- |

**GOTO <étiquette> :** Permet de se déplacer (faire un branchement) dans le bloc

Les instructions **SELECT, INSERT, DELETE, UPDATE** peuvent être utilisées dans un bloc

Elles peuvent utiliser des variables du programme mais attention les types doivent être compatibles et il faut utiliser des noms de variables différents.

Il est possible d’affecter le retour d’une requête qui contient une seule valeur dans une variable avec **SELECT … INTO**.

Exemple : Sauvegarde dans la relation Résultat le contenu de la division de 7324 par 9.

| DECLARE RESTE NUMBER := 7324 ; BEGIN WHILE RESTE >=9 LOOP RESTE := RESTE-9 ; END LOOP ; INSERT INTO RESULTAT VALUES (reste, 'reste division 7324 par 9') ; END ; / |
| --- |

**Les curseurs :** établir la transition entre l'univers BD et celui des langages procéduraux classiques. Permettent de pouvoir manipuler un à un tous les tuples retournés par une requête. Un curseur est défini dans la partie déclarative d'un bloc PL/SQL par une requête d'interrogation en SQL (sa structure correspond aux attributs du SELECT), en suivant la syntaxe suivante :

CURSOR <nom\_curseur> IS <requête\_SQL> ;

Exemple :

| DECLARE une\_variable NUMBER(4); CURSOR C\_pilote IS SELECT Plnum, Plnom |
| --- |

**Gestion des curseurs : Dans le corps du bloc entre BEGIN et END**

**OPEN** : exécute la requête de définition du curseur et alloue la place mémoire nécessaire. Le curseur peut alors être perçu comme une suite d'enregistrements.

**CLOSE** : désactive le curseur et libère la place mémoire. Le curseur est alors perçu comme un ensemble indéfini.

**FETCH INTO** : ramène le prochain enregistrement du curseur et renseigne les différentes variables réceptrices.

**Les attributs des curseurs** sont des propriétés booléennes prédéfinies des curseurs.

**<nom\_curseur>%NOTFOUND**

est à vrai si l'ordre FETCH ne retourne aucun enregistrement.

**<nom\_curseur>%FOUND**

est à vrai si l'ordre FETCH retourne un enregistrement.

**<nom\_curseur>%ISOPEN**

est à vrai si le curseur est ouvert

**<nom\_curseur>%ROWCOUNT**

retourne le nombre de tuples qui ont été accédés via le curseur (0

avant le 1er fetch, puis 1, puis 2 ...).

Exemple avec %ISOPEN :

| IF NOT lecurseur%ISOPEN THEN OPEN lecurseur; END IF; |
| --- |

Exemple avec %FOUND :

| OPEN lecurseur; LOOP FETCH lecurseur INTO variable1,variable2; EXIT WHEN NOT lecurseur%FOUND; END LOOP; CLOSE lecurseur; |
| --- |

Exemple avec %NOTFOUND :

| OPEN lecurseur; LOOP FETCH lecurseur INTO variable1,variable2; EXIT WHEN lecurseur%NOTFOUND; END LOOP; CLOSE lecurseur; |
| --- |

Exemple dans un while :

| OPEN lecurseur; FETCH lecurseur INTO variable1,variable2; WHILE lecurseur%FOUND LOOP ... FETCH lecurseur INTO variable1,variable2; END LOOP; CLOSE lecurseur; |
| --- |

Il est possible de passer un paramètre à un curseur :

| DECLARE CURSOR lecurseur (un\_car CHARACTER) IS SELECT att1,att2 FROM table WHERE att3 = un\_car; BEGIN OPEN lecurseur('a'); |
| --- |

**Les exceptions :** IF <condieon>THEN RAISE<nom\_excepeon>; END IF ;

– **NO\_DATA\_FOUND** : déclenchée si une requête ne rend aucun résultat ;

– **ZERO\_DIVIDE** : déclenchée s'il y a tentative de division par 0 ;

– **DUP\_VAL\_ON\_INDEX** : déclenchée lors d'une tentative d'insertion d'une valeur dupliquée pour un attribut sur lequel est défini un index primaire ;

– **INVALID\_NUMBER** : déclenchée si une incompatibilité pour un type numérique est détectée.

– **INVALID\_CURSOR** déclenchée par exemple dans le cas d’accès à un curseur non ouvert.

**Le traitement des exceptions se fait dans la partie EXCEPTION du bloc PL/SQL par** :

WHEN <nom\_exception> THEN [BEGIN] <liste\_instructions>

[END] ;

Ou

WHEN OTHERS THEN <liste\_instructions>

Exemple :

| BEGIN SELECT COUNT(\*) INTO nb\_vols FROM VOL; ... IF numero > 10000 THEN RAISE impossible ; ... EXCEPTION WHEN impossible THEN numero := 0 ; WHEN OTHERS THEN numero := 100 ; END ; |
| --- |

**Les modules stockés :** Un module stocké est un programme rangé dans la base de données et être ainsi réutilisables et partageables (autorisation). Ces programmes peuvent être appelés à tout moment par un client et seront exécutés sur le serveur. Il est possible de définir des procédures ou des fonctions.

**Une procédure :**

| CREATE[OR REPLACE] PROCEDURE nom\_procedure /\* Déclaration des paramètres \*/ (var\_entree IN type, var\_sortie OUT type, var\_entrée\_sortie IN OUT type) IS /\* Déclaration des variables locales\*/ var\_locale type; BEGIN <liste instructions> [EXCEPTION ...] END ; |
| --- |

**Une fonction :**

| CREATE [OR REPLACE] FUNCTION nom\_fonction /\* Déclaration des paramètres \*/ (var\_entree IN type, ...) RETURN type IS /\* Déclaration des variables locales\*/ var\_locale type; BEGIN <liste\_instructions> RETURN (var\_locale) ; [EXCEPTION ...] END ; |
| --- |

Exemple création d’une procédure puis appel de cette dernière dans un bloc :

| CREATE OR REPLACE PROCEDURE nom\_pil ( numero IN PILOTE.Plnum%type, nom OUT PILOTE.Plnom%type) IS BEGIN SELECT Plnom INTO nom FROM PILOTE WHERE numero = Plnum ; END; |
| --- |

Appel de la procédure :

| DECLARE LeNomPilote VARCHAR(100) ; BEGIN nom\_pil(100,leNomPilote) ; -- appel de la procedure dbms\_output.put\_line(leNomPilote) ; -- affichage END; |
| --- |

Pour pouvoir afficher du texte à l’écran utilisation du package **DBMS\_OUTPUT** : au niveau du prompt SQLPlus, exécuter l’instruction suivante : **SET SERVEUROUTPUT ON**

L’instruction pour afficher du texte ou le contenu d’une variable est : **DBMS\_OUTPUT.PUT\_LINE('texte' || variable) ;**

**D’autres fonctions d’affichage :**

**dbms\_output.enable** (autorise l’affichage)

**dbms\_output.disable** (interdit l’affichage)

**dbms\_output.put\_line** (affiche la chaîne et passe à la ligne)

**dbms\_output.new\_line** (passe à la ligne)

Exemple :

| SET SERVEROUTPUT ON; DECLARE LeNomPilote VARCHAR(100) ; BEGIN nom\_pil(100,leNomPilote) ; -- appel de la procedure dbms\_output.put\_line(leNomPilote) ; -- affichage END; |
| --- |

## Triggers :

Un déclencheur est un traitement (sous forme de bloc PL/SQL) qui s’exécute automatiquement en réponse à un événement. Il y a deux types de triggers :

* Triggers base de données
* Triggers d’application

Syntaxe d’un Trigger :

| CREATE [OR REPLACE ] TRIGGER nom\_trigger {BEFORE | AFTER | INSTEAD OF } {INSERT [OR] | UPDATE [OR] | DELETE} [OF col\_name] ON table\_name [REFERENCING OLD AS o NEW AS n] [FOR EACH ROW] WHEN (condition) DECLARE Declaration-statements BEGIN Executable-statements EXCEPTION Exception-handling-statements END; |
| --- |

**DATE et SYSDATE :**

CREATE TABLE TESTDATE (LADATE DATE);

SELECT LADATE FROM TESTDATE;

**TO\_CHAR** permet de convertir les dates :

**SELECT** TO\_CHAR(LADATE, 'YYYY/MM/DD') **AS** UNEDATE **FROM** TESTDATE;

**TO\_DATE** (chaîne, '<format>') Opération inverse : conversion d’une chaîne en format DATE

**INSERT INTO** TESTDATE **VALUES**

(TO\_DATE('2016/APR/02', 'yyyy/mm/dd');

• Où format est le même que TO\_CHAR

**SYSDATE** permet de connaître la date système

• Peut être utilisé directement dans les triggers

• Par contre pour afficher la date système il faut

utiliser une relation DUAL :

SELECT TO\_CHAR (SYSDATE, 'Jour DD-Mon-YYYY

HH24') AS " Date Courante " FROM DUAL;

Exemple :

| CREATE OR REPLACE TRIGGER ctrl\_mise\_a\_jour\_employe BEFORE INSERT OR DELETE OR UPDATE ON EMPLOYES DECLARE MESSAGE EXCEPTION; BEGIN IF (TO\_CHAR(SYSDATE,'DY')= 'SAM' OR TO\_CHAR(SYSDATE,'DY')= 'DIM') THEN RAISE MESSAGE; END IF; EXCEPTION WHEN MESSAGE THEN RAISE\_APPLICATION\_ERROR(-20324,'pas de mise à jour en fin de semaine'); END; |
| --- |

Le nom d’un trigger doit être unique dans un même schéma. Même s’il peut avoir le même nom qu’un objet de la base, il est préférable d’éviter les conflits.

**BEFORE** : Le traitement est exécuté avant l’ordre LMD qui l’a déclenché.

**AFTER** : Le traitement est exécuté après l’ordre LMD qui l’a déclenché.

Remarque : les triggers **AFTER** row sont plus efficaces que les triggers **BEFORE** row car ils ne nécessitent qu’une seule lecture des données.

**INSTEAD OF** : Le traitement est exécuté en lieu et place de l’exécution de l’ordre LMD qui l’a déclenché.

*Indique quel ordre SQL déclenche le traitement :*

* INSERT
* UPDATE
* DELETE

**Table :**

– La définition précise la table associée au trigger, une et une seule table

– Pas de vue (voir INSTEAD OF)

INSTEAD OF :

| CREATE OR REPLACE TRIGGER insert\_les\_clients INSTEAD OF INSERT ON les\_clients FOR EACH ROW BEGIN INSERT INTO CLIENT (num\_client,nom,prenom) VALUES (seq\_client.nextval,:new.nom,:new.prenom) ; END; |
| --- |

Le type d’un trigger détermine quand oracle déclenche le trigger, combien de fois le traitement doit s’exécuter suite à l'événement qui l’a déclenché. Ce type est défini par BEFORE, AFTER, FOR EACH ROW.

**Les deux types de triggers :**

* Les triggers lignes : se déclenchent individuellement pour chaque ligne de la table affectée par le trigger
* Les triggers globaux : se déclenchent qu’une fois en début ou fin de transaction.

Pour spécifier un trigger ligne : FOR EACH ROW.

Pas d’accès aux valeurs nouvelles et anciennes : :new ou :ol.

Il est possible d’ajouter une restriction sur les lignes via une expression logique SQL : c'est la clause **WHEN** :

* Cette expression est évaluée pour **chaque ligne affectée par le trigger**
* Le trigger n'est déclenché sur une ligne **que si l'expression WHEN est vérifiée** pour ceIe ligne
* L'expression logique **ne peut pas contenir une sous requête WHEN** (new.empno>0)

Le corps du trigger : Il est défini sous forme d’un bloc PL/SQL. Il est exécuté si l’instruction de déclenchement se produit et si la clause de restriction WHEN, le cas échéant, est évaluée à vrai. Les corps d’un trigger ligne et d’un trigger global sont différents.

Il est possible dans un trigger en ligne d’accéder à la nouvelle valeur et à l’ancienne.

* Si l’instruction est INSERT, seule la nouvelle valeur à du sens et inversement pour DELETE, seule l’ancienne a un sens.

**:new = la nouvelle**

**: old = l’ancienne**

IF :new.salaire > :old.salaire THEN …

**REFERENCING :** Si une table s’appelle new ou old, il est possible d’utiliser REFERENCING pour éviter les ambiguïtés.

| CREATE TRIGGER nomtrigger BEFORE UPDATE ON new REFERENCING new AS autrenew FOR EACH ROW BEGIN :autrenew.colon1:= TO\_CHAR(:autrenew.colon2); END; |
| --- |

Quand un trigger comporte plusieurs instructions de déclenchement (INSERT OR DELETE OR UPDATE), on peut utiliser des prédicats conditionnels (INSERTING, DELETING et UPDATING) pour exécuter des blocs de code spécifiques pour chaque instruction de déclenchement :

| CREATE TRIGGER ... BEFORE INSERT OR UPDATE ON employe ....... FOR EACH ROW BEGIN ...... IF INSERTING THEN ....... END IF; IF UPDATING THEN ........ END IF; ...... END; |
| --- |

On peut avoir au maximum un trigger de chacun des types suivants pour chaque table :

BEFORE UPDATE row

BEFORE DELETE row

BEFORE INSERT statement

BEFORE INSERT row

…

**Les instructions autorisées :**

* Autorisées : les instructions du LMD
* Interdites : les instructions du LDD et les instructions de contrôle des transactions (ROLLBACK, COMMIT)

**Suppression :** d’un trigger = DROP TRIGGER nom\_trigger

**Activation :** Un trigger est activé par défaut.

**Désactivation d’un trigger :**  ALTER TRIGGER nomtrigger DISABLE;

**Pour désactiver tous les triggers associés à une table :** ALTER TABLE nomtable DISABLE ALL TRIGGERS;

**Pour activer un trigger :** ALTER TRIGGER nomtrigger ENABLE;

**Pour activer tous les triggers associés à une table :** ALTER TABLE nomtable ENABLE ALL TRIGGERS;

**Metabase** : Tables USER\_TRIGGERS, ALL\_TRIGGERS et DBA\_TRIGGERS.

**Procédure spécifique :**

*raise\_application\_error (error\_number,error\_message)*

* error\_number doit être un entier compris entre -20000 et -20999
* error\_message doit être une chaîne de 500 caractères maximum.
* Quand cette procédure est appelée, elle termine le trigger, défait la transaction (ROLLBACK), renvoie un numéro d'erreur défini par l'utilisateur et un message à l'application
* Si une erreur se produit pendant l'exécution d'un trigger, toutes les mises à jour produites par le trigger ainsi que par l'instruction qui l'a déclenché sont défaites

| CREATE OR REPLACE TRIGGER secure\_emp BEFORE INSERT ON EMP BEGIN IF (TO\_CHAR (SYSDATE,'DY') IN ('SAM', 'DIM')) OR (TO\_CHAR(SYSDATE,'HH24') NOT BETWEEN '08' AND '18' THEN RAISE\_APPLICATION\_ERROR (-20500, 'Vous ne pouvez utiliser la table EMP que pendant les heures normales.'); END IF; END; / |
| --- |

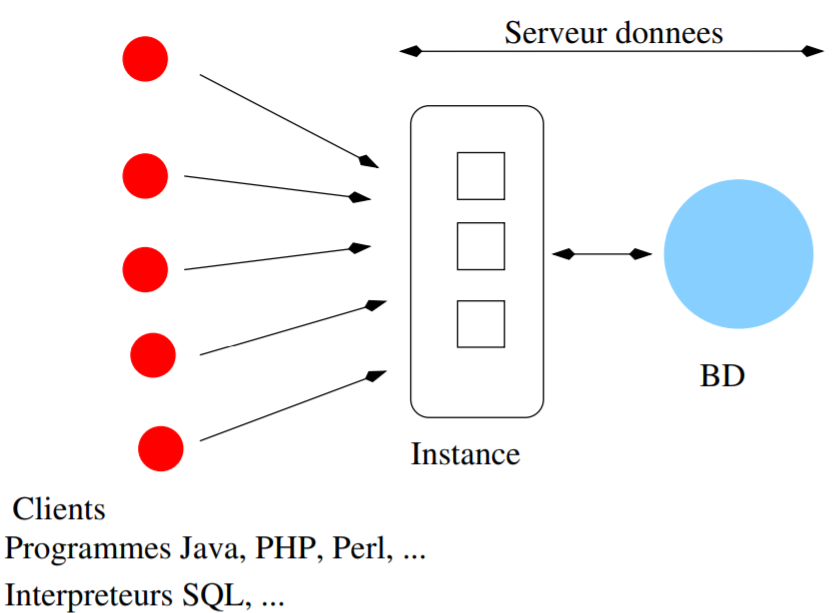
## Transactions :

**Notions importantes :**

Accès concurrent : accès simultané à une ressource (bd, table, tuple) qui peuvent aboutir à des conflits.

Session : Période délimitée dans le temps pendant laquelle un client entre en communication avec un serveur de données. Session est vue ici comme une collection de transactions.

Connexion : gestion de l’ouverture de la session - souvent associée à un mécanisme d’identification.



**Concept de la transaction :** Unité de traitement séquentiel (séquence cohérente d’actions), exécutée pour le compte d’un usager, qui appliquée à une bd

cohérente, restitue une bd cohérente.

Les opérations de la transaction doivent être soit exécutées en bloc, soit annulées en bloc (tout ou rien) =⇒ se doter d’un début et d’une fin de transaction :

Marquer le début (implicite ou explicite)

* au premier ordre SQL (ouverture de la session)
* après la fin d’une transaction (validation ou annulation)
* ordre : start transaction, begin transaction ou set transaction ....

Marquer la fin (implicite ou explicite) fin explicite d’une transaction à l'aide des commandes

COMMIT (validation des actions) et ROLLBACK (annulation des actions)

Exemple :

| CREATE TABLE Compte (numC integer primary key, typeC varchar(10), solde float); INSERT INTO Compte VALUES (2, 'courant', -200); INSERT INTO Compte VALUES (3, 'courant', 500); INSERT INTO Compte VALUES (4, 'courant', 100.50); COMMIT; UPDATE Compte SET solde = solde + 100 WHERE numC=3; ROLLBACK; DELETE FROM Compte WHERE numC=3; ALTER TABLE Compte ADD numAg integer ; |
| --- |

Gérer les transactions : s’assurer qu’elles possèdent les propriétés ACID.

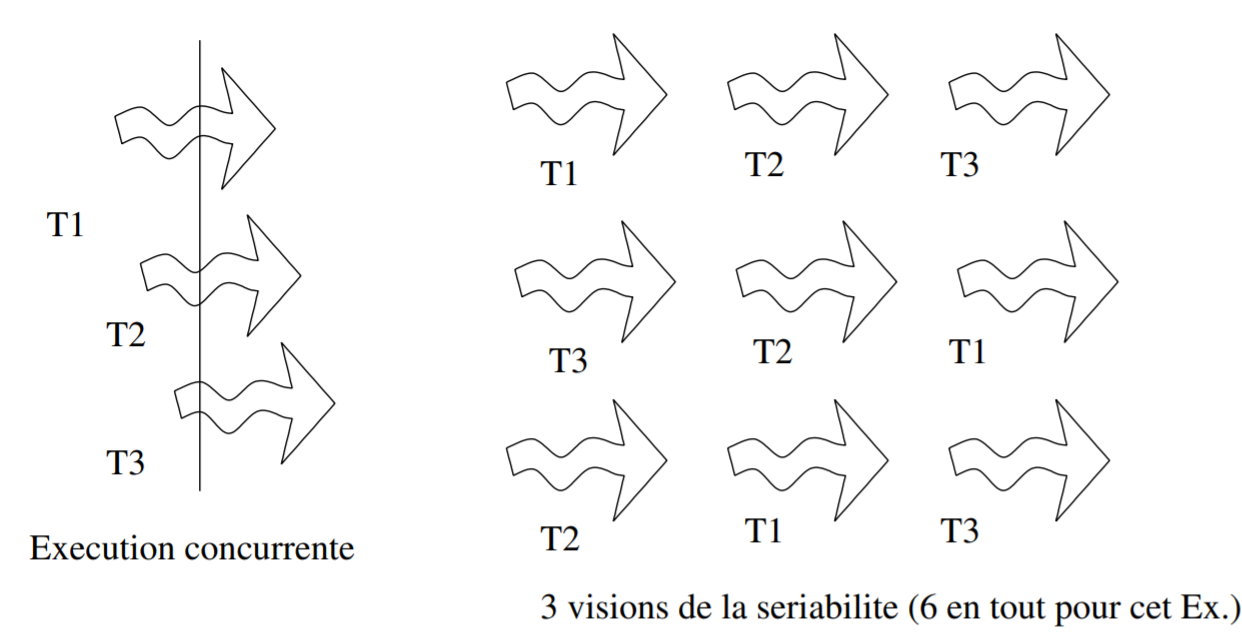
* Atomicité : lors d’une exécution d’une transaction, toutes ses actions sont exécutées ou bien aucune ne l’est.
* Cohérence : les modifications apportées à la b.d lors d’une transaction doivent être valides cad respecter les contraintes d’intégrité.
* Isolation : chaque transaction est isolée, pour s’affranchir des incohérences lors d’exécutions concurrentes
* Durabilité ou Permanence : les effets d’une transaction, qui s’est exécutée correctement, doivent survivre à une panne

La validation à deux phases suppose l’existence d’une mémoire stable, dans laquelle au point de validation, les nouvelles valeurs seront enregistrées.

* importance du point de validation : calculs pour accepter ou rejeter la transaction finissante (certification). segments d’annulation, mémoire redo-log après le point de validation, la transaction sera visible (au bout d’un certain temps) par les autres.
* transaction vivante : avant le point de validation
* transaction validée : après le point de validation

**Transactions sérialisables :**

Une exécution d’un ensemble de transactions est sérialisable ssi elle est équivalente à une exécution séquentielle (ou en série) de transactions. Quand les transactions sont arbitraires, la sériabilité est la seule à pouvoir assurer un entrelacement correct.



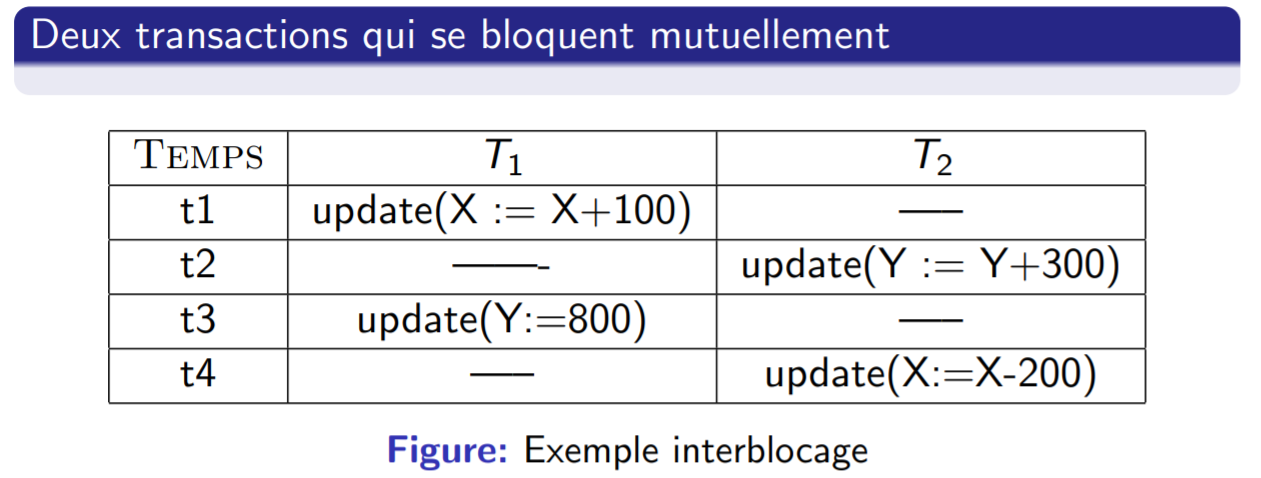
**Les verrous :**

Isoler un élément dans une transaction en le verrouillant (lock). Les verrous sont définis par deux opérations : Verrouillage à deux phases (2PL) :

* verrouiller(A) (lock(A)): cette opération oblige toute transaction à attendre le déverrouillage de l’élément A si elle a besoin de cet élément
* déverrouiller(A) (unlock(A)): la transaction effectuant cette opération libère le verrou qu’elle avait obtenu sur A et permet à une autre transaction candidate, en attente, de poser, à son tour, un verrou.

**Inconvénient des verrous :**

* **la famine** : lorqu’un verrou est relˆaché sur A, le système choisit parmi les transactions candidates en attente : ordre d’entrée dans une file d’attente
* **l’interblocage (deadlock)** : Cette situation se présente lorsqu’un ensemble de transactions attendent mutuellement le déverrouillage d’éléments actuellement verrouillés par des transactions de cet ensemble.



La pose de verrous dégrade les performances du système et impose des temps d’attente =⇒ : limiter les impacts des effets en donnant le choix sur l’objet à verrouiller =⇒ notion de granularité du verrou. La BD est découpée en granules. Les verrous peuvent porter sur

ces granules en fonction de la configuration du système fixée par l’administrateur.

Réalisation : la sériabilité impose aux transactions que tous les verrouillages précèdent tous les déverrouillages. **Les transactions sont dites à deux phases** : une phase d’acquisition des verrous puis une phase de libération (à la validation ou à l’annulation). Aucun granule ne reste verrouillé après la fin d’une transaction. En complément au protocole à deux phases, deux types de verrous sont distingués :

* **verrou partagé ou lâche** (lecture) noté S (Shared)
* **verrou exclusif ou bloquant** (écriture) noté X (eXclusive)

Synthèse des problèmes liés aux accès concurrents :

* Perte de mise à jour
* Lecture impropre (dirty read)
* Lecture de données non validées
* Lecture de donnés incohérentes
* Lecture non reproductible (non repeatable read)
* Lecture de données fantômes (phantom)

**Les degrés d’isolement d’une relation :**

De + en + performant mais de + en + contraignant (en terme de verrous posés)

* **read-uncommitted** : transaction (avec actions en écriture) sans besoin de validation pour ˆetre visible par les autres transactions
* **read-committed** : transaction (avec actions en écriture) devant ˆetre validée pour ˆetre visible par les autres transactions
* **repeatable-read** : transaction (incluant des actions en lecture et écriture) devant ˆetre validée pour ˆetre visible par les autres transactions
* **serializable** : transaction isolée : insensible aux changements intermédiaires des autres

**Points de reprise :**

Découper les transactions longues et introduire des points de reprise (marqueurs) à partir desquels il est possible de remonter en cas de problème.

Au point de validation, les effets d’une transaction doivent être conservés sur la base en toute circonstance =⇒ fichiers journaux qui conservent la trace des transactions successives. Lors d’une transaction qui effectue une mise à jour sur la base : la

base passe d’un ancien état à un nouvel état et le journal conserve l’identification des éléments modifiés, leur ancienne valeur et leur nouvelle valeur.

**Les différents ordre à attribuer :**

* **GRANT** : GRANT [privilege|privilege list|ALL|EXECUTE] ON [object] TO [schema] [WITH GRANT OPTION]; Exemple : GRANT INSERT, UPDATE ON Compte TO user1;
* **REVOKE** : REVOKE [privilege|privilege list|ALL|EXECUTE] ON [object] FROM [schema]; Exemple : REVOKE ALL ON Compte FROM public; ou bien : REVOKE INSERT, UPDATE ON Compte FROM user1;

**Renseignements sur les privilèges accordés :**

desc user\_tab\_privs