Bases de Données Avancées

UML et SQL 2/3

Thierry Hamon

Bureau H202 - Institut Galilée
Tél. : 33 1.48.38.35.53
Bureau 150 – LIM&BIO – EA 3969
Université Paris 13 - UFR Léonard de Vinci
74, rue Marcel Cachin, F-93017 Bobigny cedex
Tél. : 33 1.48.38.73.07, Fax. : 33 1.48.38.73.55
thierry.hamon@univ-paris13.fr
http://www-limbio.smbh.univ-paris13.fr/membres/hamon/BDA-20112012

INFO2 – BDA

Ingénieurs sup Galilée

Plan

De UML à SQL 2/3, Objet-Relationnel, Orienté-Objet

- Introduction
- De UML à SQL2
  (du conceptuel au relationnel étendu – objet-relationnel)
- De UML à SQL3
  (du conceptuel à l'orienté objet)
- Conclusion

Conception, Développement, Utilisation, Administration

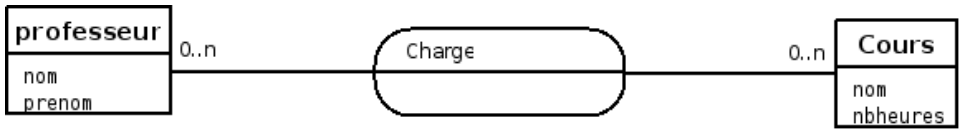
1. Etape conceptuelle : Conception et Modélisation de bases de données
2. Etape logique : Implantation d'une base de données
3. Etape physique :
4. Logiciels (SGBD, Interfaces, ...) & Matériels

Modélisation de Bases de Données

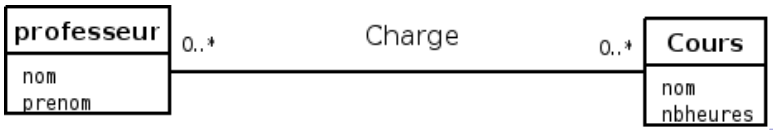
- Phase d'analyse : définition d'un schéma conceptuel
- Schéma Conceptuel de Données (SCD) : selon le formalisme utilisé,

  ensemble d'Entités et d'Associations
  ou
  ensemble de Classes

- Formalisme EA, ER



- Formalisme UML :



Modélisation de Bases de Données

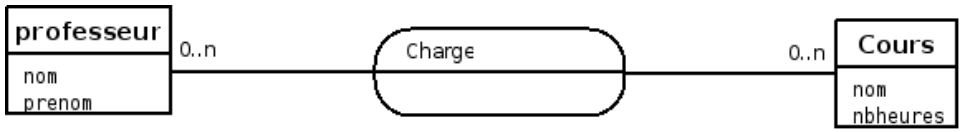
Différents formalismes de modélisation de schémas conceptuels de BD :

- Formalisme EA, ER, EER
  - Modèle Entité-Association (*Entity-Relationship Model*)
  - Modèle Entité-Association Etendu (*Extended Entity-Relationship Model*
- Formalisme UML (*Unified Modelling Language*)

Modèle Entité-Association

Rappel

- Entité : tout concept concret ou abstrait individualisable
- Classe ou type d'entités : regroupement d'entités de même nature (niveau générique)
- Association : relation liant plusieurs entités
- Classe ou type d'associations) : regroupement d'associations présentant les mêmes caractéristiques



Modèle entité-association étendu

- Modèle entité-association : jeu de concept réduit mais suffisant pour la modélisation de problèmes simples (ou peu complexes)
- Modèle Entité-association étendu : Modélisation plus précise et plus expressive de problèmes complexes et de grande taille

  Introduction de mécanismes d'abstraction

  - de classification
  - d'héritage
  - d'agrégation

Types faibles

Type d'entités ou d'associations *faibles* :

- existence d'une instance subordonnée à l'existence d'un autre type d'entité ou d'association

Classification

- Regroupement d'entités dans des classes en fonction de propriétés communes
- Possibilité de classer un objet dans plusieurs classes

Exemple :

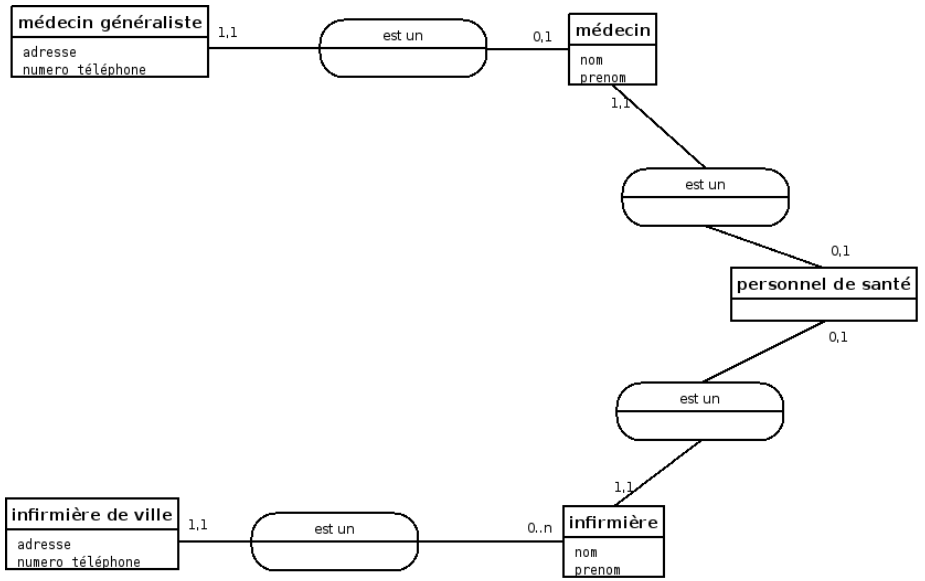
- Livre électronique : fichier électronique, et livre
- Autocar : véhicule de transport en commun, véhicule à moteur à explosion

Héritage

Spécialisation - Généralisation

Un type d' entité A est une spécialisation d'un autre type d'entité B si

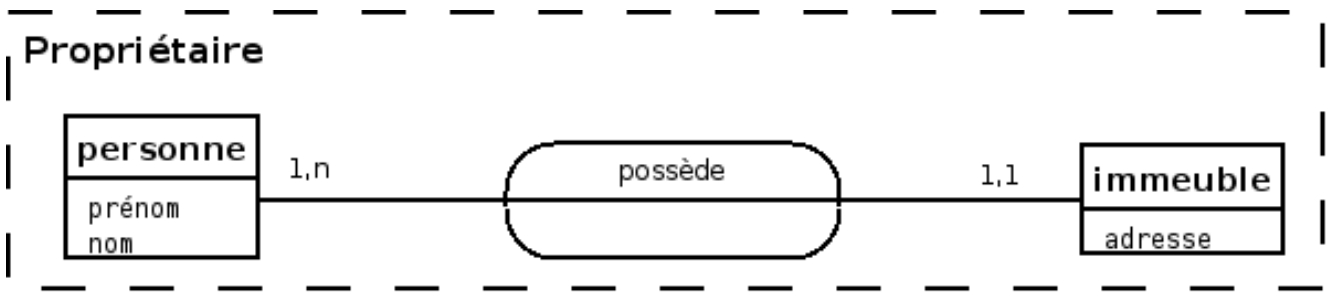
- chaque entité de A est une entité de B
- Une seule entité (au plus) de B est associé à une entité de A



Agrégation

Description de types d'entités complexes

Un type d'associations entre types d'entités est considéré comme un nouveau type d'entités



Modélisation de Bases de Données

Les deux formalismes E/R et UML sont très proches / « équivalents »

Entité/Association	→	UML
Entité		Objet
Type d'entité		Classe
Relation		Objet
Type d'association		Classe
Attribut/Propriété		Propriété
Rôle / Label		Rôle
		Méthode

Domaine	Contrainte de domaine
Clé	Contrainte de clé
Contrainte	Contrainte
Cardinalité	Multiplicité/Cardinalité
`0,1 1,1 0,n 1,n a,b a,a`	`0..1 1 0..* 1..* a..b a`
Diagramme E/A	Diagramme de Classe UML

De EA à SQL

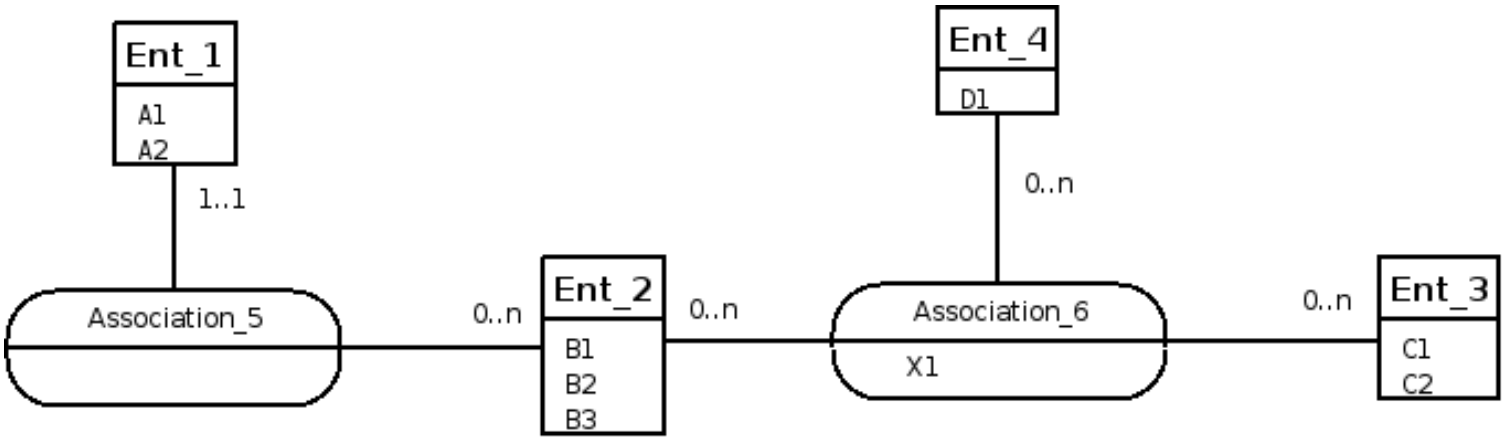
Objectifs :

- Implantation d'un schéma conceptuel (SCD) dans un BD relationnelle
- Exploitation du SCD par le SGBD et les modules de programmation

→ Transformation dans un schéma relationnel : Schéma Logique de Données (SLD)

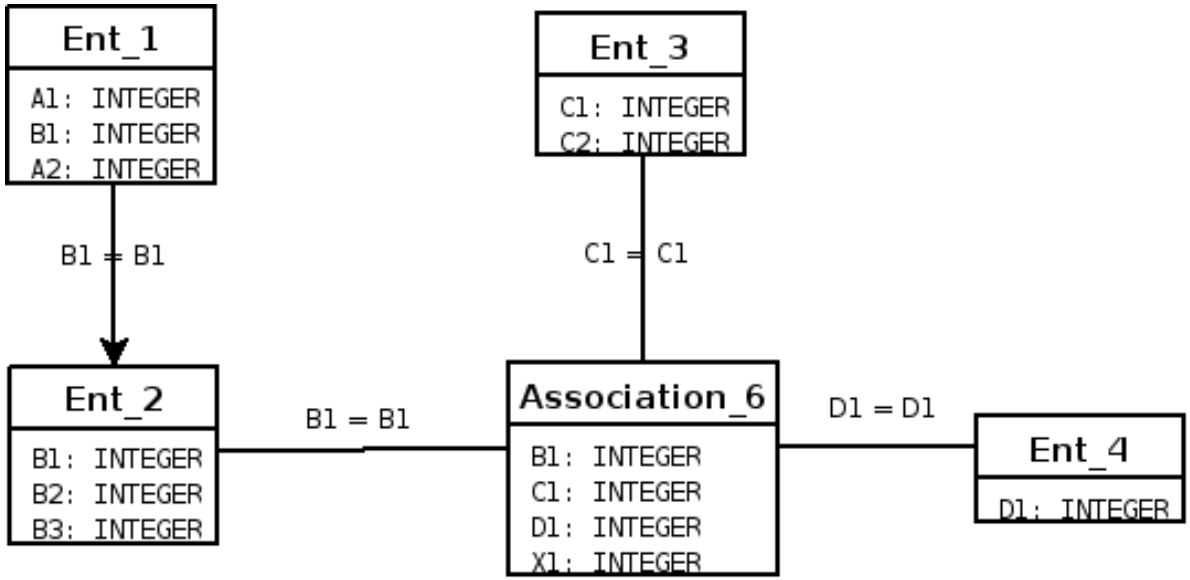
De EA à SQL

Exemple de schéma conceptuel EA (SCD)



De EA à SQL

Exemple de schéma logique (schéma relationnel) SLD



R1_ENT_2 (B1, B2, B3)
R2_ENT_1 (A1, A2, B1 *)
R3_ENT_3 (C1, C2)
R4_ENT_4 (D1)
R5_ASSOC_6 (B1*, C1*, D1*, X1)

Règles de passage

du modèle Entité-Association au modèle Relationnel

- Tout type d'entité E est traduit en une relation R
  - La clé primaire de R est l'identifiant de E
  - Les attributs de R sont ceux de E.
- Tout type d'association est traduit :
  - en une clé étrangère dans une relation existante si la cardinalité est du type `1,1` ou `0,1`
  - en une nouvelle relation si aucune cardinalité n'est du type `1,1` ou `0,1` (elles sont toutes du type `0,n` ou `1,n`)

Plusieurs algorithmes sont possibles selon l'interprétation de la cardinalité minimale égale à 0.

De UML à SQL

- Traduction des associations binaires
- Traduction des associations binaires récursives

- Traduction des associations n-aires ($n > 2$)

- Traduction des associations d'héritage
- Traduction des contraintes d'héritage

- Traduction des associations d'agrégation

- Traduction des contraintes d'intégrité fonctionnelles (contraintes : Partition, Exclusion, Totalité, Simultanéité, Inclusion)

Intégrité des données (1)

Les SGBD prennent en compte l'intégrité des données définies via

- la déclaration de contraintes (constraints)
- la programmation de
  - fonctions (functions)
  - de procédures (procedures) cataloguées
  - de paquetages (packages)
  - de déclencheurs (triggers)

Le principe étant d'assurer la cohérence de la base après chaque mise à jour par les commandes `insert`, `update` *ou* `delete`

Ingénieurs sup Galilée

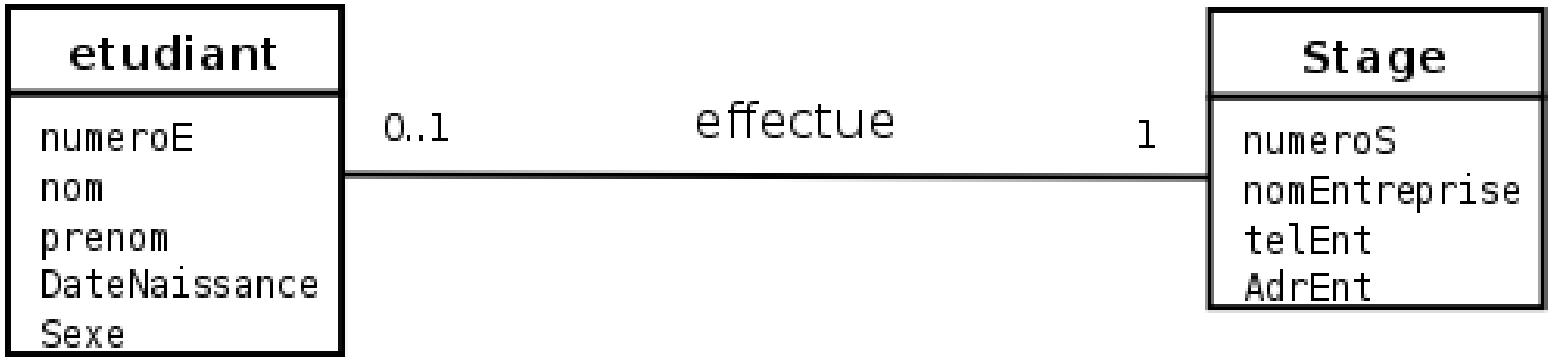
Intégrité des données (2)

Notation utilisée concernant les noms des contraintes :

- la contrainte clé primaire d'une table se nomme `pk_table`
- la contrainte clé étrangère d'une table se nomme `fk_table1_colonne_table2`
- la contrainte de validité d'une colonne se nomme `ck_table_colonne`
- la contrainte de type non nulle sur une colonne se nomme `nn_table_colonne`
- la contrainte de type unique sur une colonne se nomme `unique_table_colonne`

Traduction des associations binaires (1)

Association du type 1-1



REM *** Un stage est effectué par au plus un étudiant

```
create table STAGE
(   NUMEROS number(7),
    NOMENTREPRISE varchar(40),
    TELENT varchar(15),
    ADRENT varchar(50),
    constraint PK_STAGE primary key (NUMEROS)
);
```

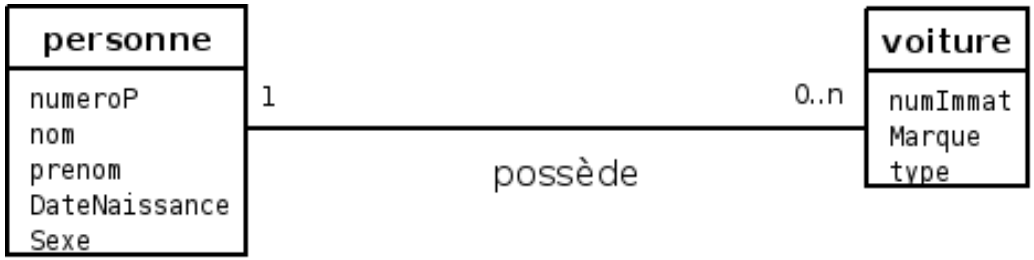
Traduction des associations binaires (2)

Association du type 1-1

```
REM *** Un étudiant effectue obligatoirement un stage unique

create table ETUDIANT
( NUMEROE number(7),
  NOM varchar(10),
  PRENOM varchar(10),
  DATENAISSANCE date,
  SEXE char(1),
  NUMEROS number(7),
  constraint PK_ETUDIANT primary key (NUMEROE),
  constraint FK_ETUDIANT_NUMEROS_STAGE foreign key (NUMEROS)
  references STAGE(NUMEROS),
  constraint CK_ETUDIANT_SEXE check (SEXE in ('M', 'F')),

--
  constraint NN_ETUDIANT_NUMEROS check (NUMEROS is not null),
  constraint UNIQUE_ETUDIANT_NUMEROS unique (NUMEROS)
);
```

Traduction des associations binaires (1)

Association du type 1-N




```
REM *** Une personne peut posséder plusieurs voitures

create table PERSONNE
(
   NUMEROP number(7),
   NOM varchar(10),
   PRENOM varchar(10),
   DATENAISSANCE date,
   SEXE char(1),
   constraint PK_PERSONNE primary key (NUMEROP)
);
```

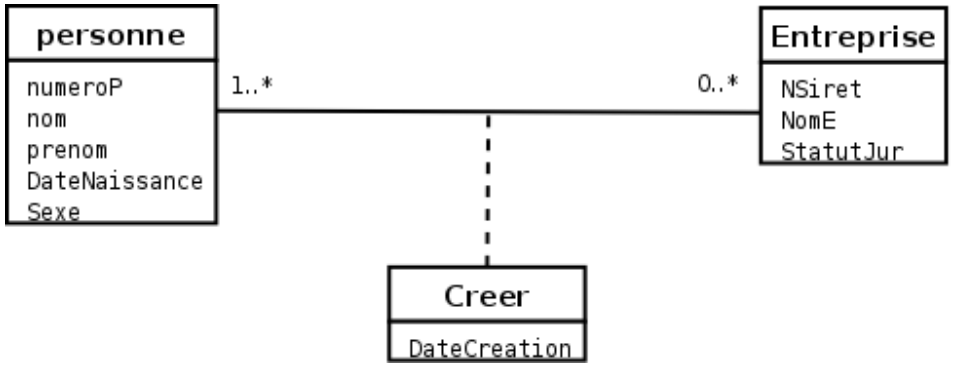
Traduction des associations binaires (2)

Association du type 1-N

```
REM *** Une voiture est obligatoirement possédée par une personne

create table VOITURE
(
   NUMIMMAT varchar(15),
   MARQUE varchar(20),
   TYPE varchar(30),
   NUMEROP number(7),
    constraint PK_VOITURE primary key (NUMIMMAT),
    constraint FK_VOITURE_NUMEROP_PERSONNE foreign key (NUMEROP)
                        references PERSONNE(NUMEROP),

--
    constraint NN_VOITURE_NUMEROP check (NUMEROP is not null)
);
```

Traduction des associations binaires (1)

Association du type N-N




```
REM *** Une personne peut créer plusieurs entreprises

create table PERSONNE
(
   NUMEROP number(7),
   NOMP varchar(10),
   PRENOM varchar(10),
   DATENAISSANCE date,
   SEXE char(1),
    constraint PK_PERSONNE primary key (NUMEROP)
);
```

Traduction des associations binaires (2)

Association du type N-N

```
REM *** Une entreprise doit être créée
REM     par une ou plusieurs personnes

create table ENTREPRISE
(
   NSIRET varchar(20),
   NOME varchar(20),
   STATUTJUR varchar(10),
   constraint PK_ENTREPRISE primary key (NSIRET)
);
```

Traduction des associations binaires (3)

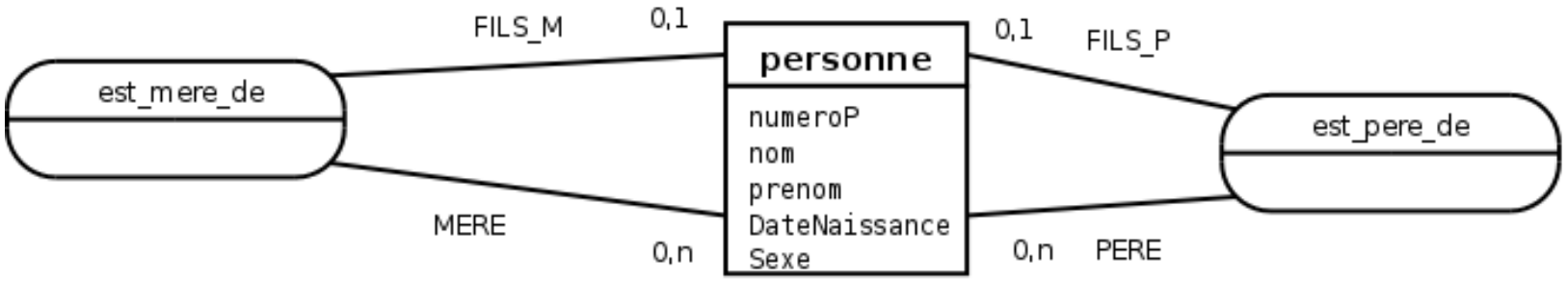
Association du type N-N

```
create table CREER
(
   NUMEROP number(7),
   NSIRET varchar(20),
   DATECREATION date,
   constraint PK_CREER primary key (NUMEROP, NSIRET),
   constraint FK_CREER_NUMEROP_PERSONNE
                  foreign key (NUMEROP)
                  references PERSONNE(NUMEROP),
   constraint FK_CREER_NSIRET_ENTREPRISE
                  foreign key (NSIRET)
                  references ENTREPRISE(NSIRET)
);
```

La cardinalité minimale de l'association créer pourra être testée par l'intermédiaire d'une procédure PL/SQL

Traduction des associations binaires (1)

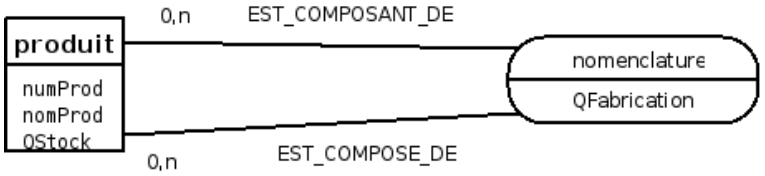
Association du type Réflexif/Récursif, UML ∼ EA




```
create table PERSONNES
(
  NUMERO number(7), NOM varchar(15), PRENOM varchar(15),
  DATENAISSANCE date, SEXE char(1), PERE number(7),
  MERE number(7),
   constraint PK_PERSONNES primary key (NUMERO),
   constraint FK_PERSONNES_PERE_PERSONNES foreign key(PERE)
              references PERSONNES,
   constraint FK_PERSONNES_MERE_PERSONNES foreign key(MERE)
              references PERSONNES,
   constraint CK_SEXE_PERSONNES check (SEXE in ('M', 'F'))
);
```

Traduction des associations binaires (2)

Association du type Réflexif/Récursif, UML ∼ EA




```
create table PRODUITS
(
  NUMERO number(3), NOM varchar(15), QSTOCK number(5),
   constraint PK_PRODUITS primary key (NUMERO),
   constraint CK_QSTOCK_PRODUITS check (QSTOCK >= 0)
);
```

Traduction des associations binaires (3)

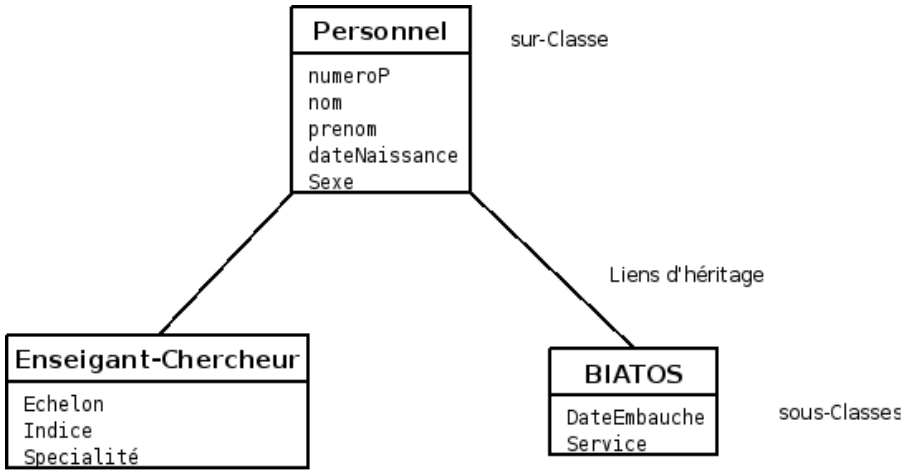
Association du type Réflexif/Récursif, UML ∼ EA

```
create table NOMENCLATURE
(
  COMPOSER number(3), COMPOSANT number(3), QFABR number(5),
   constraint PK_NOMENCLATURE primary key (COMPOSER, COMPOSANT),
   constraint FK_NOMENCL_COMPOSER_PRODUITS foreign key(COMPOSER)
              references PRODUITS,
   constraint FK_NOMENCL_COMPOSANT_PRODUITS foreign key(COMPOSANT)
              references PRODUITS,
   constraint CK_QFABR_NOMENCLATURE check (QFABR >= 0)
);
```

Traduction des associations d'héritage

Traduction des contraintes d'héritage

Gestion du personnel dans une université



Associations d'héritage dans UML (1)

- Recensement des différents cas d'héritage en fonction des instances
- Modélisation des différents héritages, dans le formalisme UML, à l'aide des contraintes
  - partition
  - exclusion
  - totalité

Associations d'héritage dans UML (2)

Expression des cas d'héritage à l'aide de

- couverture
- disjonction

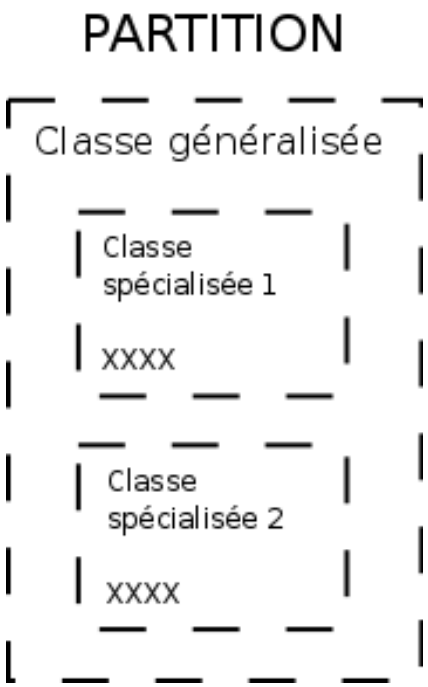
d'instances dans une population donnée

Quatre type de contraintes sont recensés :

- partition
- totalité
- exclusion
- absence de contrainte

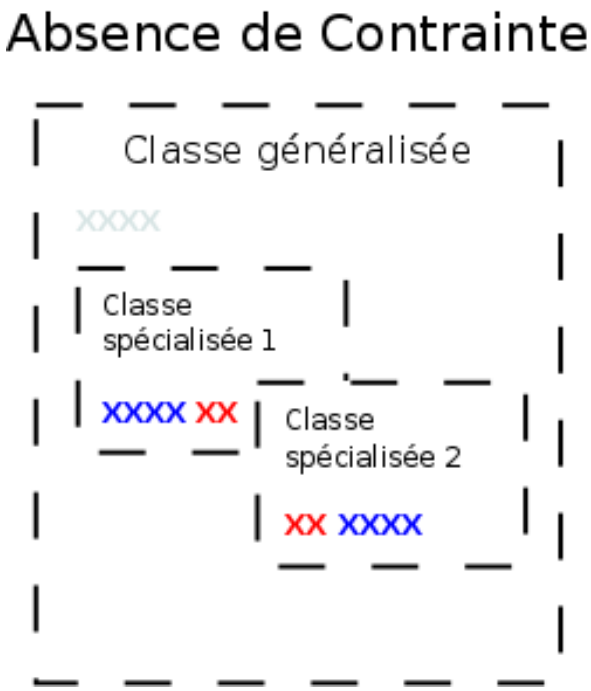
Contraintes d'héritages PARTITION et TOTALITE

- Disjonction & Couverture → Partition
- Non-Disjonction & Couverture → Totalité



Contraintes d'héritages EXCLUSION et ABSENCE DE CONTRAINTE

- Disjonction & Non-Couverture → Exclusion
- Non-Disjonction & Non-Couverture → Absence de contrainte



Exemple (1)

Gestion du personnel dans une université

- Couverture + Disjonction → Partition

  Personnel (P) est égal à l'Union de Enseignant (EC) et de BIATOS (B) et l'Intersection de EC et de B est Vide

- Couverture + Non-Disjonction → Totalité

  Personnel (P) est égal à l'Union de Enseignant (EC) et de BIATOS (B) et l'Intersection de EC et de B n'est pas Vide

Exemple (2)

Gestion du personnel dans une université

- Non-Couverture + Disjonction → Exclusion

  L'Union de Enseignant (EC) et de BIATOS (B) est incluse dans P et l'Intersection de EC et de B est Vide

- Non-Couverture + Non-Disjonction → Absence de contraintes

  L'Union de Enseignant (EC) et de BIATOS (B) est incluse dans P et l'Intersection de EC et de B n'est pas Vide

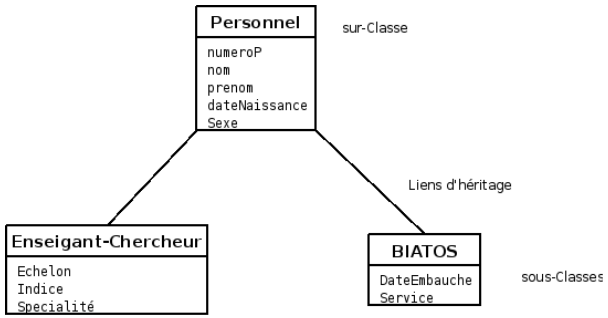
Transformation des associations d'héritage

Traduction d'une association d'héritage en fonction des contraintes de l'association d'héritage
$\rightarrow$ 3 familles de décomposition :

- Décomposition par distinction
- Décomposition descendante (*push-down*)
- Décomposition ascendante (*push-up*)

Décomposition par distinction

- Transformation de chaque sous-classe en une relation
- Migration de la clé primaire de la sur-classe dans la ou les relations issues des sous-classes
- La clé primaire de la sur-classe devient à la fois clé primaire et clé étrangère



Distinction

PERSONNEL(Numéro, Nom, Prénom, DateNaissance, Sexe)

ENSEIGNANT(Numéro*, Echelon, Indice, Spécialité)

BIATOS(Numéro*, DateEmbauche, Service)

Ingénieurs sup Galilée

Décomposition descendante

Deux cas possibles selon la contrainte d'héritage :

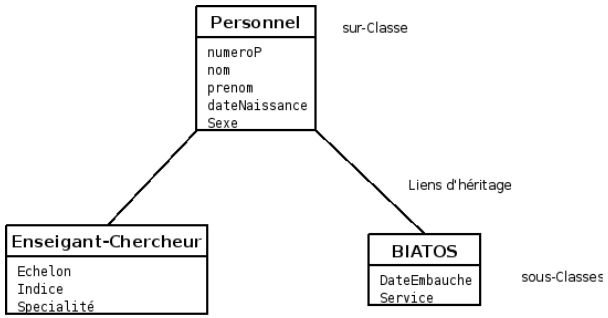
- Contrainte de totalité ou de partition sur l'association :
  Possibilité de ne pas traduire la relation issue de la sur-classe
  $\rightarrow$ Migration de tous les attributs dans la ou les relations issues de la ou des sous-classes
- Sinon : Migration de tous les attributs dans la ou les relations issues de la ou des sous-classes
  $\rightarrow$ Duplication des données

Décomposition descendante

Exemple

Contrainte de partition :

- Aucun personnel ne peut être à la fois enseignant et BIATOS
- Il n'existe pas non plus un personnel n'étant ni enseignant ni biatos.



Descendante

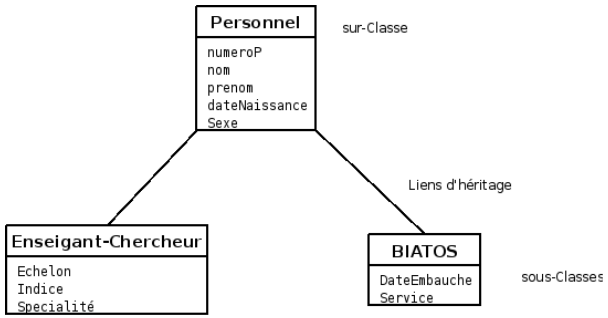
ENSEIGNANT(Numéro, Nom, Prénom, DateNaissance, Sexe, Echelon, Indice, Spécialité)

BIATOS(Numéro, Nom, Prénom, DateNaissance, Sexe, DateEmbauche, Service)

Décomposition ascendante

- Suppression de la ou les relations issues de la ou des sous-classes
- Migration des attributs dans la relation issue de la sur-classe

Exemple : (absence de contrainte)



Ascendante

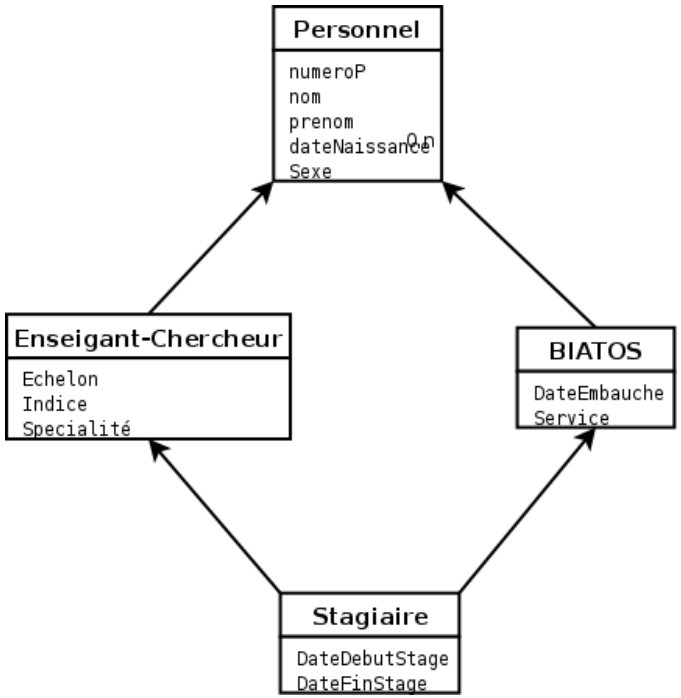
PERSONNEL(Numéro, Nom, Prénom, DateNaissance, Sexe, Echelon, Indice, Spécialité, DateEmbauche, Service)

Transformation des associations d'héritage multiple

Mêmes règles ; plusieurs possibilités

Exemple : (décomposition ascendante)

Contrainte d'exclusion sur `enseignant` et `BIATOS`



PERSONNEL(Numéro, Nom, Prénom, DateNaissance, Sexe)

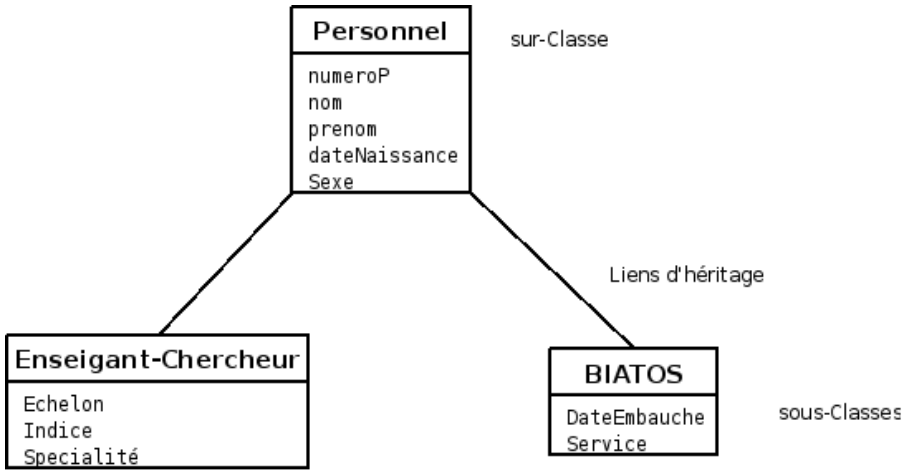
ENSEIGNANT (Numéro*, Echelon, Indice, Spécialité, DateDébutStage, DateFinStage)

BIATOS(Numéro*, DateEmbauche, Service, DateDébutStage, DateFinStage)

Transformation des associations d'héritage en SQL 2

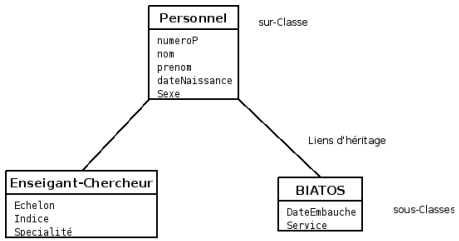
Exemple

Gestion du personnel dans une université



Transformation des associations d'héritage

Décomposition par distinction



Distinction

PERSONNEL(Numéro, Nom, Prénom, DateNaissance, Sexe)

ENSEIGNANT(Numéro*, Echelon, Indice, Spécialité)

BIATOS(Numéro*, DateEmbauche, Service)

```
REM *** Un personnel à l'Université

create table PERSONNEL
( NUMERO number(7),
  NOM varchar(10),
  PRENOM varchar(10),
  DATENAISSANCE date,
  SEXE char(1),
  constraint PK_PERSONNEL primary key (NUMERO),
  constraint CK_SEXE_PERSONNEL check (SEXE in ('M', 'F'))
);
```

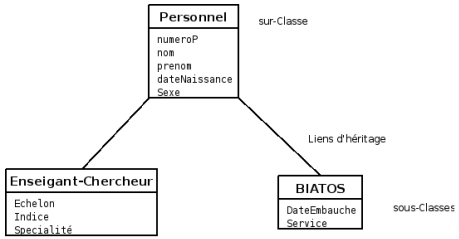
Transformation des associations d'héritage

Décomposition par distinction

```
REM *** Personnel enseignant

create table ENSEIGNANT
( NUMERO number(7), ECHELON number(2),
  INDICE number(5), SPECIALITE varchar(20),
  constraint PK_ENSEIGNANT primary key (NUMERO),
  constraint FK_ENS_PERS foreign key (NUMERO)
                  references PERSONNEL
);

REM *** Personnel BIATOS (Ing, Adm, Tech, Ouv, Serv)

create table BIATOS
( NUMERO number(7), DATEEMBAUCHE date,
  SERVICE varchar(20),
  constraint PK_BIATOS primary key (NUMERO),
  constraint FK_BIATOS_PERS foreign key (NUMERO)
                  references PERSONNEL
);
```

Transformation des associations d'héritage

Décomposition descendante



Descendante

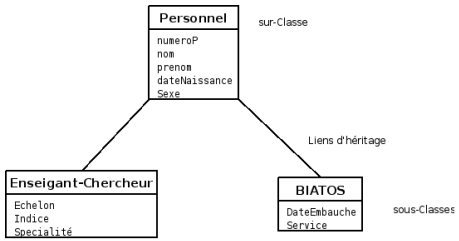
ENSEIGNANT(Numéro, Nom, Prénom, DateNaissance, Sexe, Echelon, Indice, Spécialité)

BIATOS(Numéro, Nom, Prénom, DateNaissance, Sexe, DateEmbauche, Service)

```
REM *** Personnel enseignant

create table ENSEIGNANT
( NUMERO number(7), NOM varchar(10),
  PRENOM varchar(10), DATENAISSANCE date,
  SEXE char(1), ECHELON number(2),
  INDICE number(5), SPECIALITE varchar(20),
  constraint CK_SEXE_ ENSEIGNANT check
                              (SEXE in ('M', 'F'))
  constraint PK_ENSEIGNANT primary key (NUMERO)
);
```

Transformation des associations d'héritage

Décomposition descendante



Descendante

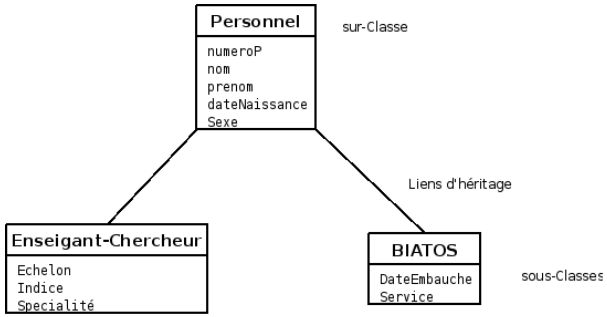
ENSEIGNANT(Numéro, Nom, Prénom, DateNaissance, Sexe, Echelon, Indice, Spécialité)

BIATOS(Numéro, Nom, Prénom, DateNaissance, Sexe, DateEmbauche, Service)

```
REM *** Personnel BIATOS

create table BIATOS
( NUMERO number(7), NOM varchar(10), PRENOM varchar(10),
  DATENAISSANCE date, SEXE char(1), DATEEMBAUCHE date,
  SERVICE varchar(20),
   constraint CK_SEXE_ BIATOS check (SEXE in ('M', 'F'))
   constraint PK_BIATOS primary key (NUMERO)
);
```

Transformation des associations d'héritage

Décomposition ascendante



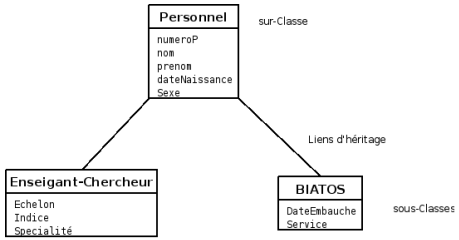
Ascendante

PERSONNEL(Numéro, Nom, Prénom, DateNaissance, Sexe, Echelon, Indice, Spécialité, DateEmbauche, Service)

```
REM *** Personnel

create table PERSONNEL
( NUMERO number(7), NOM varchar(10), PRENOM varchar(10),
  DATENAISSANCE date, SEXE char(1), ECHELON number(2),
  INDICE number(5), SPECIALITE varchar(20),
  DATEEMBAUCHE date, SERVICE varchar(20),
  constraint CK_SEXE_ PERSONNEL check (SEXE in ('M', 'F'))
  constraint PK_ PERSONNEL primary key (NUMERO)
);
```

Traduction des contraintes d'héritage

Décomposition par distinction



- → *Contrainte de partition*
- Contrainte de totalité
- Contrainte d'exclusion
- Sans Contrainte

Contraintes d'héritage :

- (Contrainte A) Il n'existe pas de personnel à la fois enseignant et BIATOS
- (Contrainte B) Il n'existe pas de personnel ni enseignant ni BIATOS

Traduction des contraintes d'héritage

Décomposition par distinction

Implémentation de la contrainte A : 2 déclencheurs

```
REM *** Déclencheur sur ENSEIGNANT

create or replace trigger TRIG_ENSEIGNANT
before insert or update of NUMERO on ENSEIGNANT
for each row
declare
    num number;
begin
    select NUMERO INTO num
    from BIATOS where NUMERO = :new.NUMERO;
    raise_application_error(-20001, 'Le personnel '||
                to_char(num)||' est déjà BIATOS !!!');
exception
    when no_data_found then null;
end;
/
```

Traduction des contraintes d'héritage

Décomposition par distinction

```
REM *** Déclencheur sur BIATOS

create or replace trigger TRIG_BIATOS
before insert or update of NUMERO on BIATOS
for each row
declare
   num number;
begin
   select NUMERO INTO num
   from ENSEIGNANT where NUMERO = :new.NUMERO;
   raise_application_error(-20001, 'Le personnel '||
      to_char(num)||' est déjà enseignant !!!');
exception
   when no_data_found then null;
end;
/
```

Traduction des contraintes d'héritage

Décomposition par distinction

Implémentation de la contrainte B :

- procédures cataloguées (Insertion, Suppression)
- déclencheurs (Modification)

```
REM *** Ajout d'un Enseignant

create or replace procedure AJOUT_ENSIGNANT
(NUM number, NOM varchar, PREN varchar, DNAIS date,
 SEXE varchar,  ECHEL number, IND number, SPEC varchar) is
begin
    insert into PERSONNEL values (NUM, NOM, PREN, DNAIS, SEXE);
    insert into ENSEIGNANT values
                (NUM, ECHEL, IND, SPEC);
end;
/
```

Traduction des contraintes d'héritage

Décomposition par distinction

```
REM *** Ajout d'un BIATOS

create or replace procedure AJOUT_BIATOS
(NUM number, NOM varchar, PREN varchar, DNAIS date,
 SEXE varchar, DEMB date, SERV varchar) is
begin
    insert into PERSONNEL values (NUM, NOM, PREN, DNAIS, SEXE);
    insert into BIATOS values (NUM, DEMB, SERV);
end;
/
```

Traduction des contraintes d'héritage

Décomposition par distinction

```
REM *** Suppression d'un Enseignant

create or replace procedure SUPPR_ENSIGNANT
(NUM number) is
begin
     delete from ENSEIGNANT where NUMERO = num ;
     delete from PERSONNEL where NUMERO = num ;
end;
/

REM *** Suppression d'un BIATOS

create or replace procedure SUPPR_BIATOS
(NUM number) is
begin
     delete from BIATOS where NUMERO = num ;
     delete from PERSONNEL where NUMERO = num ;
end;
/
```

Traduction des contraintes d'héritage

Décomposition par distinction

```
REM *** Déclencheur pour la répercussion de la modification
du numéro du PERSONNEL vers ENSIGNANT et BIATOS

create or replace trigger TRIG_ENSBIATOS
before update of NUMERO on PERSONNEL
for each row
begin
  begin
      update ENSEIGNANT
      set NUMERO = :new.NUMERO
      where NUMERO = :old.NUMERO;
  exception
      when no_data_found then null;
  end;
  update BIATOS
      set NUMERO = :new.NUMERO
  where NUMERO = :old.NUMERO;
exception
  when no_data_found then null;
end;
```

Traduction des contraintes d'héritage

Utilisation

```
REM ****** Insertions des données sous SQLPLUS
REM ****** Lancement des procédures

select 'Insertion_des_données' from dual;
PAUSE

execute AJOUT_ENSIGNANT (1, 'TRAIFOR', 'Clément',
                  '17-09-1958', 'M', 6, 780, 'BD');
execute AJOUT_ENSIGNANT (2, 'TRAIFOR', 'Clémentine',
                  '22-11-1969', 'F', 6, 780, 'IA');

execute AJOUT_BIATOS (3, 'FAITOUT', 'Alex',
    '16-10-1960', 'M', '01-01-2002', 'Commercial');

PAUSE

select * from PERSONNEL;
select * from ENSEIGNANT;
select * from BIATOS;
PAUSE
```

Traduction des contraintes d'héritage

Décomposition par distinction

→ *Contrainte de totalité*

Contraintes d'héritage :

- (Contrainte B) Il n'existe pas de personnel ni enseignant ni BIATOS
- (Contrainte C) Il peut exister un personnel à la fois enseignant et BIATOS

Implémentation :

- Contrainte B : voir ci-dessus
- Contrainte C : équivaut à ne pas programmer la contrainte A précédente
  ⇒ Pas de mise en œuvre les déclencheurs des tables `ENSEIGNANT` et `BIATOS` : `TRIG_ENSEIGNANT`, `TRIG_BIATOS`

Traduction des contraintes d'héritage

Décomposition par distinction

→ *Contrainte d'exclusion*

Contraintes d'héritage :

- (Contrainte A) Il n'existe pas de personnel à la fois enseignant et BIATOS
- (Contrainte D) Il peut exister un personnel ni enseignant ni BIATOS

Implémentation :

- Contrainte A: voir ci-dessus
- Contrainte D : équivaut à ne pas programmer la contrainte B précédente
  ⇒pas de mise en œuvre les quatre procédures (ajout et suppression) et le déclencheur `TRIG_ENSBIATOS`

Traduction des contraintes d'héritage

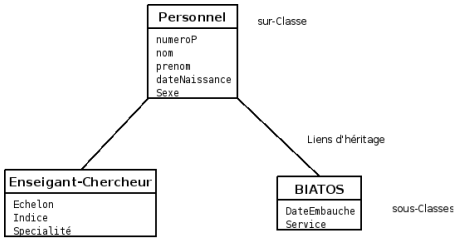
Décomposition par distinction

$\rightarrow$ *Sans Contrainte*

Aucune contrainte n'est à programmer !

Traduction des contraintes d'héritage

Décomposition descendante



Descendante

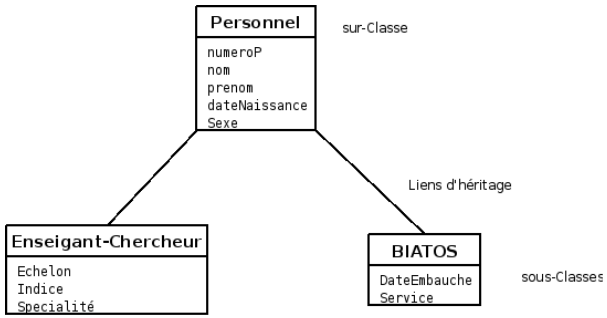
ENSEIGNANT(Numéro, Nom, Prénom, DateNaissance, Sexe, Echelon, Indice, Spécialité)

BIATOS(Numéro, Nom, Prénom, DateNaissance, Sexe, DateEmbauche, Service)

- Contrainte de partition ? → *aucun personnel ne peut être à la fois enseignant et iatos et il n'existe pas non plus un personnel n'étant ni enseignant ni iatos*
- Contrainte de totalité ?
- Contrainte d'exclusion ? → *il faudrait la table personnel pour les personnels non enseignant et non BIATOS*
- Sans contrainte !

Traduction des contraintes d'héritage

Décomposition ascendante



- → *Contrainte de partition*
- Contrainte de totalité
- Contrainte d'exclusion
- Sans Contrainte

Contraintes d'héritage :

- (Contrainte A) Il n'existe pas de personnel à la fois enseignant et BIATOS
- (Contrainte B) Il n'existe pas de personnel ni enseignant ni BIATOS

Traduction des contraintes d'héritage

Décomposition ascendante

Implémentation des contraintes A et B :

- au niveau de la table `personnel`
- à l'aide des contraintes de type `CHECK`

- Contrainte A :
  Vérifier que les colonnes `ECHELON`, `INDICE`, `SPECIALITE`, `DATEEMBAUCHE` et `SERVICE` ne soient pas toutes initialisées
- Contrainte B :
  Vérifier que les colonnes `ECHELON`, `INDICE`, `SPECIALITE`, `DATEEMBAUCHE` et `SERVICE` ne soient pas toutes nulles

Traduction des contraintes d'héritage

Décomposition ascendante

```
REM **** CONTRAINTE A

alter table PERSONNEL
  add constraint CK_CONTRAINTE_A
      check (
          (ECHELON is null and INDICE is null and
           SPECIALITE is null)
        or (DATEEMBAUCHE is null and SERVICE is null)
      );

REM **** CONTRAINTE B

alter table PERSONNEL
  add constraint CK_CONTRAINTE_B
      check (
            (ECHELON is not null or INDICE is not null or
             SPECIALITE is not null)
        or  (DATEEMBAUCHE is not null or SERVICE is not null)
      );
```

Traduction des contraintes d'héritage

Décomposition ascendante

→ *Contrainte de totalité*

Contraintes d'héritage :

- (Contrainte B) Il n'existe pas de personnel ni enseignant ni BIATOS
- (Contrainte C) Il peut exister un personnel à la fois enseignant et BIATOS

Traduction des contraintes d'héritage

Décomposition ascendante

→ *Contrainte de totalité*

- Contrainte B : voir ci-dessus
- Contrainte C : Suppression ou désactiver la contrainte A précédente (**DROP CONSTRAINT** ou DISABLE **CONSTRAINT**)

  - **DROP CONSTRAINT** : en cas de réactivation de la contrainte, il est nécessaire de la recréer (**ADD CONSTRAINT**)
  - DISABLE **CONSTRAINT** : en cas de réactivation de la contrainte, il faut simplement la réactiver avec la requête ENABLE **CONSTRAINT**

Traduction des contraintes d'héritage

Décomposition ascendante

→ *Contrainte de totalité*

```
REM La contrainte C revient à faire la
REM Désactivation de la CONTRAINTE A

alter table PERSONNEL
  disable constraint CK_CONTRAINTE_A;
```

Traduction des contraintes d'héritage

Décomposition ascendante

→ *Contrainte d'exclusion*

Contrainte d'héritage :

- Contrainte A → Réactivation de la contrainte A en supprimant au préalable les tuples ne répondant pas à cette contrainte
- Non-contrainte B → Désactivation de la contrainte B

Traduction des contraintes d'héritage

Décomposition ascendante

→ *Contrainte d'exclusion*

```
REM ***** Réactivation de la CONTRAINTE A
alter table PERSONNEL
     enable constraint CK_CONTRAINTE_A;


REM ***** Désactivation de la CONTRAINTE B
alter table PERSONNEL
     disable constraint CK_CONTRAINTE_B;
```

Traduction des contraintes d'héritage

Décomposition ascendante

$\rightarrow$ *Sans Contrainte*

Aucune contrainte de type `CHECK` n'est à programmer !

Conclusion / Bilan

Aucune des solutions ne constitue la panacée.

Il faut mesurer les performances des requêtes.

Voir aussi le type de requêtes

Transformation des associations d'héritage en SQL 3

- Héritage de types
  - Existe depuis la version 9.1 d'Oracle (novembre 2001)
  - Uniquement héritage de type
  - Pas d'héritage multiple
  - Un type peut hériter d'un seul autre type (**sur-type**)
  - Un sur-type peut permettre de définir plusieurs sous-types
  - Chaque sous-type est spécialisé par rapport au sur-type qui est dit plus général
  - Mécanisme d'héritage automatiquement répercuté au niveau des tables objet à par du moment où les types définissant les tables sont issus eux-mêmes d'une hiérarchie d'héritage
- Héritage de tables ?

Héritage de types

Définition d'un personnel à l'Université

```
--- ***        Création du type de la sur-classe
create type PERSONNEL_TYPE AS OBJECT
(
   NUMERO number(7),
   NOM varchar(10),
   PRENOM varchar(10),
   DATENAISSANCE date,
   SEXE char(1)
)
NOT FINAL  /* peut inclure des sous classes */
/
```

Héritage de types

Définition d'un enseignant

```
--- *** Création du type de la sous-classe
create type ENSEIGNANT_TYPE UNDER PERSONNEL_TYPE
(
  ECHELON number(2),
  INDICE number(5),
  SPECIALITE varchar(20)
)
FINAL
/
```

Création des tables objet et contraintes

Ci-dessous :

- Création de tables objet en fonctions des types précédemment définis
- Aucune directive ne précise l'héritage : il est induit par la hiérarchie de type existante

```
-- *** Personnel de l'université
create table PERSONNEL OF PERSONNEL_TYPE
(
   constraint PK_PERSONNEL primary key (NUMERO),
   constraint CK_SEXE_PERSONNEL check (SEXE in ('M', 'F'))
);
-- *** Personnel enseignant
create table ENSEIGNANT OF ENSEIGNANT_TYPE ;
```

IMPORTANT : les contraintes ne sont définies que dans la table `personnel`

Création des tables objet et contraintes

Illustration

NB : Les contraintes ne sont définies que dans la table `personnel`
→ On hérite d'un type

Insertion des données dans la table `personnel` :

```
insert into personnel values (1, 'B', 'F', '17-09-2004', 'M');
insert into personnel values (1, 'B', 'F', '17-09-2004', 'M');
*
ERREUR à la ligne 1 :

ORA-00001: violation de contrainte unique (FB.PK_PERSONNEL)

select * from personnel;
 NUMERO       NOM          PRENOM       DATENAISSA S
---------- ---------- ---------- ---------- -
         1 B          F          17-09-2004 M
```

Création des tables objet et contraintes

Illustration

NB : Les contraintes ne sont définies que dans la table `personnel`
$\rightarrow$ On hérite d'un type

Insertion des données dans la table enseignant :

```
insert into enseignant values (7, 'B', 'F', '17-09-2004', 'M', 2, 780, 'BD'); 1 ligne cr
insert into enseignant values (7, 'B', 'F', '17-09-2004', 'M', 2, 780, 'BD');  1 ligne c
insert into enseignant values (8, 'B', 'D', '17-10-2004', 'M', 2, 780, 'BD');  1 ligne c

select * from enseignant ;
```

NUMERO	NOM	PRENO	DATENAISSA	S	ECHELON	INDICE	SPECIALITE
7	B	F	17-09-2004	M	2	780	BD
7	B	F	17-09-2004	M	2	780	BD
8	B	D	17-10-2004	M	2	780	BD

Création des tables objet et contraintes (2)

Ci-dessous :

- Création des tables objet en fonctions des types précédemment définis
- Définition des contraintes au niveau des tables

```
-- *** Personnel de l'université
create table PERSONNEL OF PERSONNEL_TYPE
(
  constraint PK_PERSONNEL primary key (NUMERO),
  constraint CK_SEXE_PERSONNEL check (SEXE in ('M', 'F'))
);
-- *** Personnel enseignant
```

Ingénieurs sup Galilée

Transformation des associations d'héritage en SQL 3

```
create table ENSEIGNANT OF ENSEIGNANT_TYPE
(
  constraint PK_ENSEIGNANT primary key (NUMERO),
  constraint CK_SEXE_ENSEIGNANT check (SEXE in ('M', 'F'))
);
```

ATTENTION :

- Définition des contraintes, aussi dans la table enseignant
- Héritage d'un type

Création des tables objet et contraintes (2)

illustrations

Les contraintes doivent être définies aussi dans la table enseignant : héritage d'un type

```
SQL> insert into enseignant values (7, 'B', 'F', '17-09-2004', 'M', 2, 780, 'BD'); 1 lig
SQL> insert into enseignant values (7, 'B', 'F', '17-09-2004', 'M', 2, 780, 'BD');
insert into enseignant values (7, 'B', 'F', '17-09-2004', 'M', 2, 780, 'BD')
* ERREUR à la ligne 1 : ORA-00001: violation de contrainte unique (FB.PK_ENSEIGNANT)

SQL> insert into enseignant values (8, 'B', 'D', '17-10-2004', 'M', 2, 780, 'BD'); 1 lig
SQL> insert into enseignant values (9, 'B', 'D', '17-10-2004', 'K', 2, 780, 'BD');
insert into enseignant values (9, 'B', 'D', '17-10-2004', 'K', 2, 780, 'BD')
* ERREUR à la ligne 1 : ORA-02290: violation de contraintes (FB.CK_SEXE_ENSEIGNANT)
    de vérification

SQL> select * from enseignant ;
```

NUMERO	NOM	PRENO	DATENAISSA	S	ECHELON	INDICE	SPECIALITE
7	B	F	17-09-2004	M	2	780	BD
8	B	D	17-10-2004	M	2	780	BD

Transformation des associations d'héritage en SQL 3

Héritage de tables : À venir

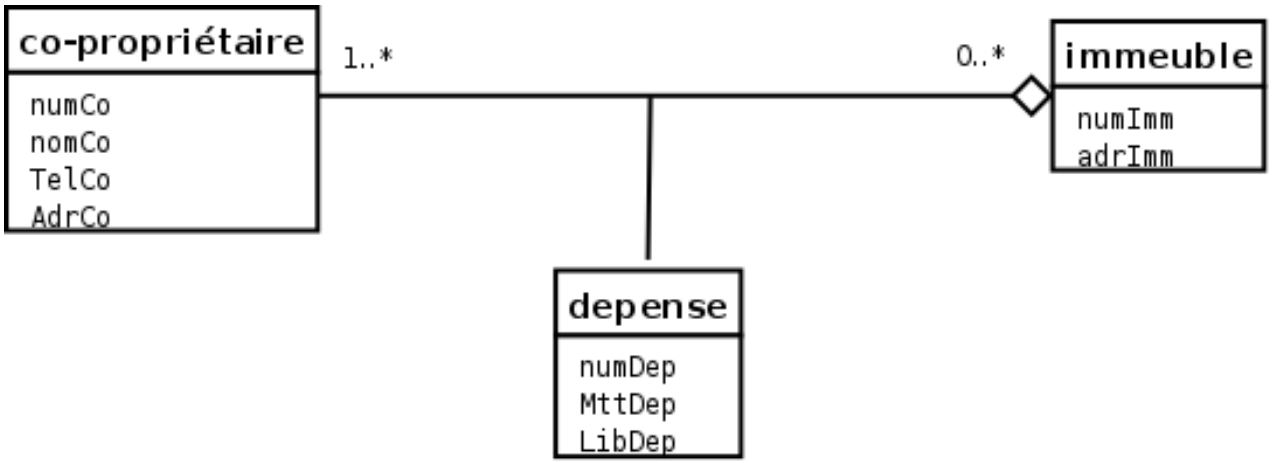
```
REM *** Un personnel à l'Université
create table PERSONNEL
(
  NUMERO number(7), NOM varchar(10),
  PRENOM varchar(10), DATENAISSANCE date,
  SEXE char(1),
   constraint PK_PERSONNEL primary key (NUMERO),
   constraint CK_SEXE_PERSONNEL check (SEXE in ('M', 'F'))
);
```

Transformation des associations d'héritage en SQL 3

Héritage de tables : À venir

```
REM *** Personnel enseignant
create table ENSEIGNANT under PERSONNEL
(
  ECHELON number(2),
  INDICE number(5),
  SPECIALITE varchar(20)
);

REM *** Personnel biatos
create table BIATOS under PERSONNEL
(
  DATEEMBAUCHE date,
  SERVICE varchar(20)
);
```

Traduction des associations d'agrégation



Traduction des associations d'agrégation

```
REM *** Un co-propriétaire peut posséder plusieurs immeubles
create table COPROPRIETAIRE
(
  NUMCO number(7),
  NOMCO varchar(10),
  TELCO varchar(15),
  ADRCO varchar(50),
   constraint PK_COPROPRIETAIRE primary key (NUMCO)
);

REM *** Un immeuble doit être possédé par un ou
REM      plusieurs copropriétaires
create table IMMEUBLE
(
  NUMIMM number(7),
  ADRIMM varchar(50),
   constraint PK_IMMEUBLE primary key (NUMIMM)
);
```

Traduction des associations d'agrégation

```
create table DEPENSE
(
  NUMCO number(7),
  NUMIMM number(7),
  DATEDEP date,
  MTTDEP number(10,2),
  LIBDEP varchar(50),
  constraint PK_DEPENSE primary key (NUMCO,NUMIMM),
  constraint FK_DEPENSE_NUMCO_COPROPR foreign key (NUMCO)
references COPROPRIETAIRE(NUMCO) on delete cascade,
  constraint FK_DEPENSE_NUMIMM_IMMEUBLE foreign key (NUMIMM)
references IMMEUBLE(NUMIMM ) on delete cascade
);
```

→ *La cardinalité minimale de l'association dépenser pourra être testée par l'intermédiaire d'une procédure PL/SQL*

Traduction des contraintes d'intégrité fonctionnelles

Contraintes : Partition, Exclusion, Totalité, Simultanéité, Inclusion

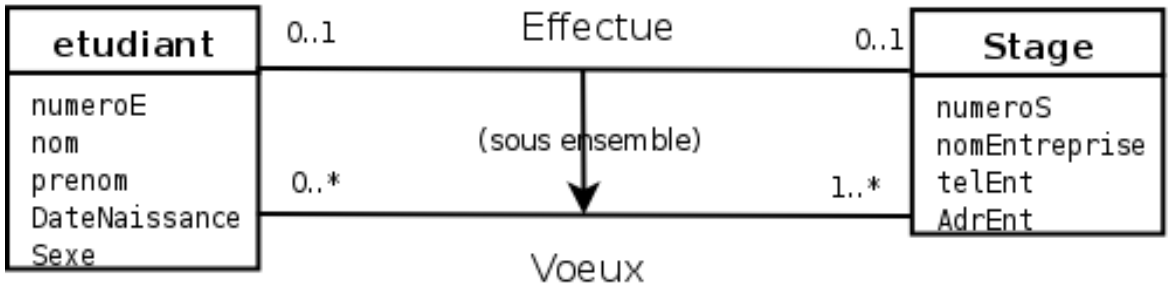
...

Toutes les contraintes peuvent être définies ou programmées via :

- la déclaration de contraintes (**constraints**)
- la programmation de
  - fonctions (**functions**)
  - procédures (**procedures**)
  - paquetages (**packages**)
  - déclencheurs (**triggers**)

  en PL/SQL ou avec un langages hôtes tels que le C, C++, Java

Contrainte d'inclusion




```
create table STAGE
(
  NUMEROS number(7),
  NOMENTREPRISE varchar(40),
  TELENT varchar(15),
  ADRENT varchar(50),
  constraint PK_STAGE primary key (NUMEROS)
);
```

Contrainte d'inclusion

Une table par classe

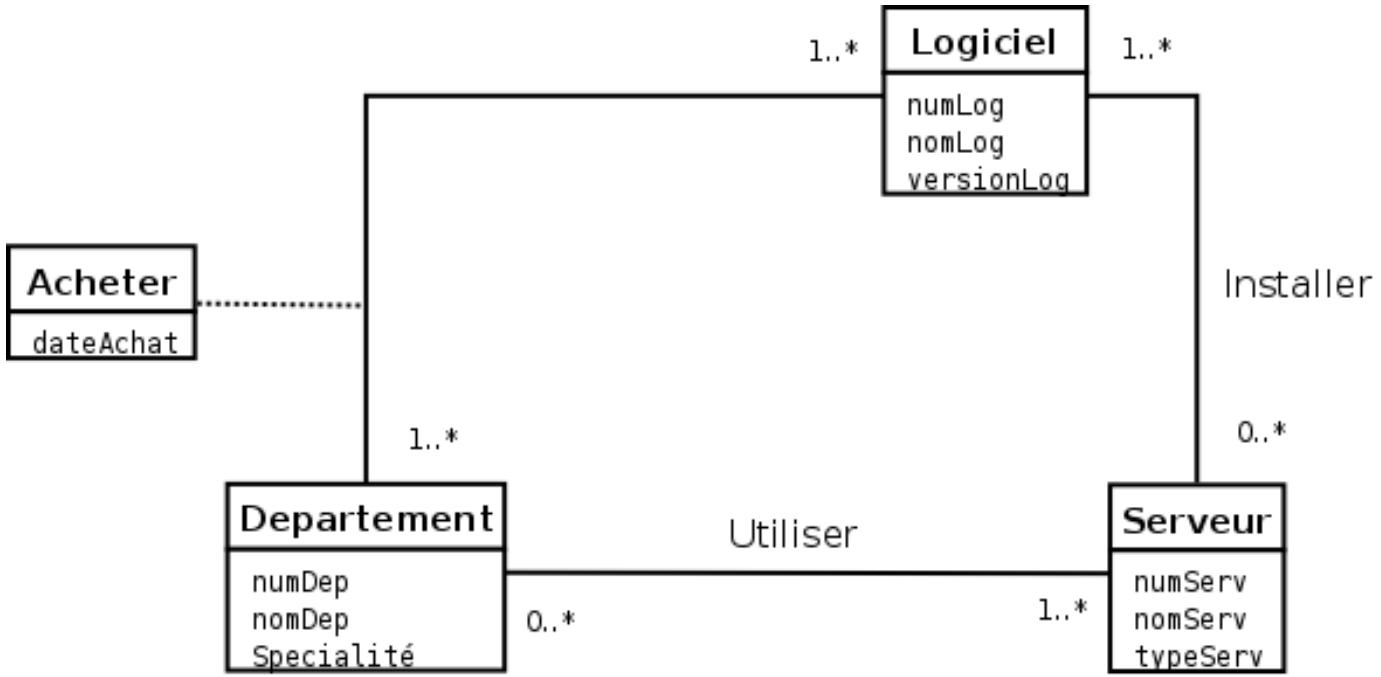
```
create table ETUDIANT
(
  NUMEROE number(7),
  NOM varchar(10),
  PRENOM varchar(10),
  DATENAISSANCE date,
  SEXE char(1),
  NUMEROS number(7),
  constraint PK_ETUDIANT primary key (NUMEROE),
  constraint FK_ETUDIANT_NUMEROS_STAGE foreign key (NUMEROS)
                                  references STAGE(NUMEROS),
  constraint CK_ETUDIANT_SEXE check (SEXE in ('M', 'F'))
);
```

Contrainte d'inclusion

```
create table VOEUX
(
  NUMEROE number(7),
  NUMEROS number(7),
   constraint PK_ VOEUX primary key (NUMEROE,NUMEROS),
   constraint FK_ VOEUX _NUMEROE_ ETUDIANT foreign key (NUMEROE)
            references ETUDIANT (NUMEROE),
   constraint FK_ VOEUX _NUMEROS_STAGE foreign key (NUMEROS)
   references STAGE(NUMEROS)
);

alter table ETUDIANT add
        constraint FK_EFFECTUER_INCLUSION_VOEUX
        foreign key (NUMEROE,NUMEROS)
        referencesVOEUX(NUMEROE,NUMEROS);
```

Contrainte d'inclusion

Contrainte d'inclusion : *Le logiciel doit être installé sur un serveur du département qui a acheté le programme.*



Un logiciel L acheté par le département D est installé sur un serveur S, destiné entre autres, à ce département

Ingénieurs sup Galilée

Contrainte d'inclusion

Une table par Classe

```
create table DEPARTEMENT
(
  NUMDEP number(7),
  NOMDEP varchar(10),
  SPECIALITE varchar(20),
  constraint PK_DEPARTEMENT primary key (NUMDEP)
);

create table LOGICIEL
(
  NUMLOG number(7),
  NOMLOG varchar(10),
  VERSIONLOG varchar(10),
  constraint PK_LOGICIEL primary key (NUMLOG)
);
```

Contrainte d'inclusion

Une table par Classe

```
create table SERVEUR
(
  NUMSERV number(7),
  NOMSERV varchar(10),
  TYPESERV varchar(10),
  constraint PK_SERVEUR primary key (NUMSERV)
);
```

Contrainte d'inclusion

Une table par Association ou par Classe-Association

```
create table ACHETER (
  NUMDEP number(7), NUMLOG number(7), DATEACHAT date,
   constraint PK_ACHETER primary key (NUMDEP,NUMLOG),
   constraint FK_ACHETER_NUMDEP_DEPARTEMENT
         foreign key (NUMDEP) references DEPARTEMENT(NUMDEP),
   constraint FK_ACHETER_NUMLOG_LOGICIEL
         foreign key (NUMLOG) references LOGICIEL(NUMLOG)
);

create table UTILISER (
  NUMDEP number(7), NUMSERV number(7),
   constraint PK_UTILISER primary key (NUMDEP,NUMSERV),
   constraint FK_UTILISER_NUMDEP_DEPARTEMENT
         foreign key (NUMDEP) references DEPARTEMENT(NUMDEP),
   constraint FK_UTILISER_NUMSERV_SERVEUR
         foreign key (NUMSERV) references SERVEUR(NUMSERV)
);
```

Contrainte d'inclusion

Une table par Association ou par Classe-Association

```
create table INSTALLER (
  NUMLOG number(7), NUMSERV number(7),
   constraint PK_INSTALLER primary key (NUMLOG,NUMSERV),
   constraint FK_INSTALLER_NUMLOG_LOGICIEL
         foreign key (NUMLOG) references LOGICIEL(NUMLOG),
   constraint FK_INSTALLER_NUMSERV_SERVEUR
         foreign key (NUMSERV) references ERVEUR(NUMSERV) );
```

Contrainte d'Inclusion

Déclencheur

Un logiciel L acheté par le département D est installé sur un serveur S, destiné entre autres, à ce département

```
create or replace trigger trig_contrainte_inclusion
  before insert on INSTALLER
  for each row
  declare
  LOGIC number(7);
  SERV number(7);
  begin
        select ACHETER.NUMLOG, UTILISER.NUMSERV into LOGIC, SERV
        from ACHETER, UTILISER
        where ACHETER.NUMDEP = UTILISER.NUMDEP and
              ACHETER.NUMLOG = :new.NUMLOG and
              UTILISER.NUMSERV = :new.NUMSERV;
  exception
        when no_data_found then
        raise_application_error(-20100,
          'Le logiciel doit être installé sur
          un serveur du département acheteur');
```

Ingénieurs sup Galilée

Programmation Objet – SQL 3

- Objet-relationnel – Objet
- Passage UML $\rightarrow$ Objet / Objet relationnel

Ingénieurs sup Galilée

Schéma relationnel / SQL2

- Schéma relationnel :

  COURS (NUM_COURS, NOMC, NBHEURES, ANNEE)

  PROFESSEURS (NUM_PROF, NOMP, SPECIALITE, DATE_ENTREE,
  DER_PROM, SALAIRE_BASE, SALAIRE_ACTUEL
)

  CHARGE(NUM_PROF*, NUM_COURS*)

Ingénieurs sup Galilée

Schéma relationnel / SQL2

- SQL2 :

```
create table COURS
( NUM_COURS     NUMBER(2)    NOT NULL,
  NOMC          VARCHAR(20)  NOT NULL,
  NBHEURES      NUMBER(2),
  ANNE          NUMBER(1),
   constraint PK_COURS primary key (NUM_COURS)
);

create table PROFESSEURS
( NUM_PROF          NUMBER(4)      NOT NULL,
  NOMP              VARCHAR2(25)   NOT NULL,
  SPECIALITE        VARCHAR2(20),
  DATE_ENTREE       DATE,
  DER_PROM          DATE,
  SALAIRE_BASE      NUMBER,
  SALAIRE_ACTUEL    NUMBER,
   constraint PK_PROFESSEURS primary key (NUM_PROF)
);

create table CHARGE
( NUM_PROF          NUMBER(4)   NOT NULL,
  NUM_COURS         NUMBER(4)   NOT NULL,
   constraint PK_CHARGE primary key (NUM_COURS,
                                     NUM_PROF)
);
```

Ingénieurs sup Galilée

Schéma relationnel / SQL2

```
alter table CHARGE
    add constraint FK_CHARGE_COURS
        foreign key  (NUM_COURS)
            references COURS (NUM_COURS);

alter table CHARGE
    add constraint FK_CHARGE_PROFESSEUR
        foreign key  (NUM_PROF)
            references PROFESSEURS (NUM_PROF);
```

Ingénieurs sup Galilée

Schéma relationnel-objet / SQL3

- Schéma relationnel-objet

COURS (NUM_COURS, NOMC, NBHEURES, ANNEE)

PROFESSEURS (
NUM_PROF, NOMP, SPECIALITE, DATE_ENTREE,
DER_PROM, SALAIRE_BASE, SALAIRE_ACTUEL,
EnsembleDe (COURS)
)

Ingénieurs sup Galilée

Schéma relationnel-objet / SQL3

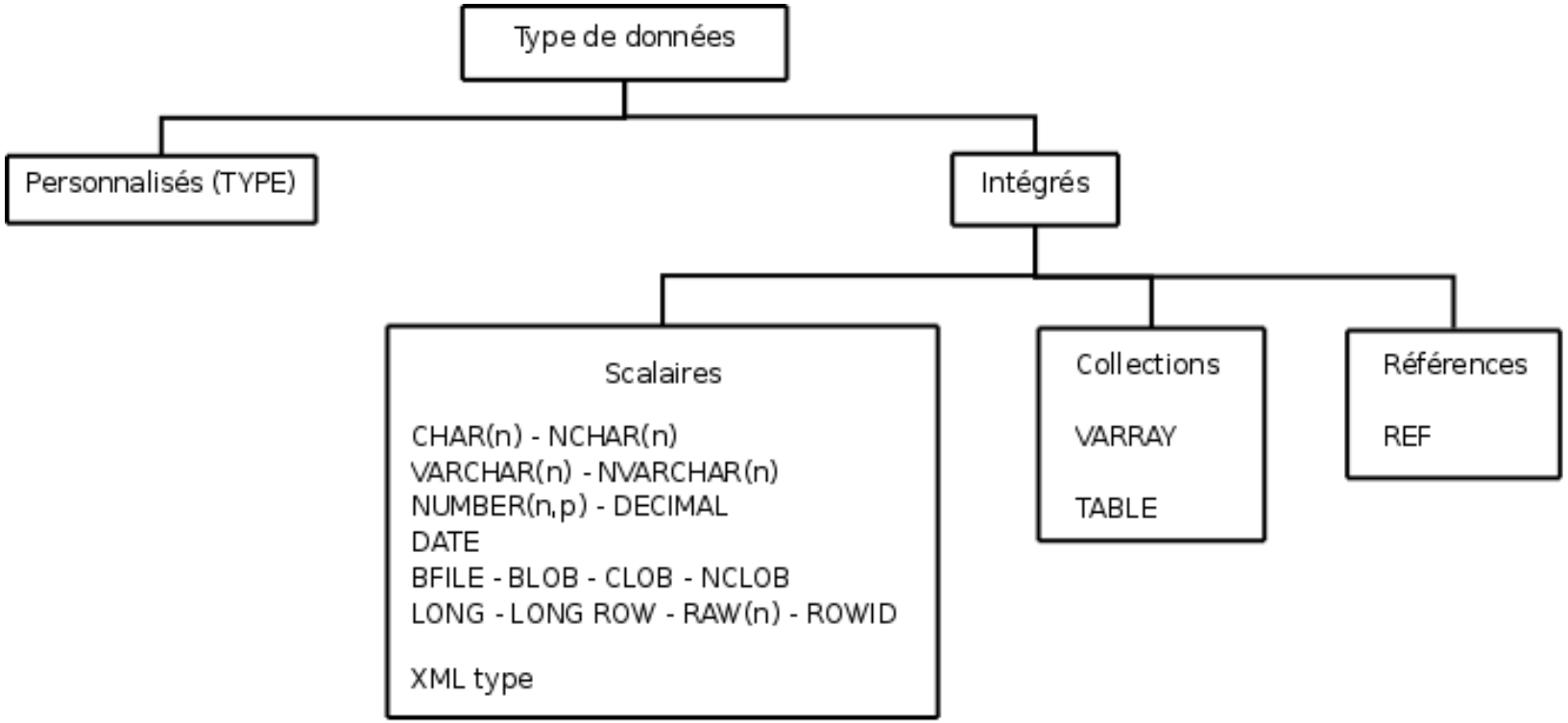
- SQL3 :

```
create type cours_type as object
( num_cours number(2), nomc varchar2(20),
   nbheures number(2), annee number(1) )
/

create  type lescours_type as table of cours_type
/

create  type professeur_type as object
( num_prof number(4), nom varchar2(25),
  specialite varchar2(20), cours lescours_type ...)
/

create table professeur of professeur_type
( primary key (num_prof) )
nested table cours store as tabemp
/
```

Types Objet

Type de données

Principaux type de données Oracle :



Types Objet

Persistance

Sous Oracle, 3 catégories d'objets :

- Objets colonne (*column objects*) : stockés en tant que colonne structurée dans une table relationnelle ;
- Objets ligne (`row objects`) : stockés en tant que ligne d'une table objet.
  - possèdent un identificateur unique appelé OID (*Object Identifier*)
  - peuvent être indexés et partitionnés
- Objets non persistants : non stockés
  - ni dans une colonne d'une table relationnelle
  - ni dans une ligne d'une table objet

  Ces objets n'existent que durant l'exécution d'un programme PL/SQL

Types Objet

- Définition de chaque objet à partir d'un type décrivant
  - une structure de données se positionnant dans une hiérarchie d'héritage
  - des méthodes
- Utilisation d'un type :
  - Construire d'autres types
  - Définir une ou plusieurs tables objet
  - Définir une colonne d'une table relationnelle
  - Construire des vues objet

Types Objet

Création d'un type

- Création

```
CREATE [OR REPLACE TYPE] schéma.nomType
                    [AS OBJECT | UNDER schéma.nomSurType]
(
  REM *** définition de la structure
  colonne1 type1, colonne2 type2, ...,


  REM *** définition du comportement
  méthode1 (paramètres1), méthode2 (paramètres2) ...
)
[[NOT] INSTANCIABLE]

REM *** positionnement dans le graphe d'héritage
[[NOT] FINAL]
/
```

Création d'un type

Directive FINAL

- Directives FINAL et NOT FINAL : positionnement d'un type dans le graphe d'héritage
- Directive NOT final : à appliquer aux types génériques
- Par défaut, tout type est FINAL
  Un type FINAL ne peut servir à définir des sous-types

Création d'un type

Directive FINAL – Exemples

```
CREATE TYPE adresse_t AS OBJECT (
 nrue NUMBER(3), rue VARCHAR(40), ville VARCHAR(30)
/

CREATE TYPE Personnel_t AS OBJECT(
 nom VARCHAR (10), prenom VARCHAR(10), adresse adresse_t))
 NOT FINAL
/

CREATE TYPE Enseignant_t AS OBJECT UNDER Personnel_t(
 Echelon NUMBER, indice NUMBER)
 FINAL
```

Création d'un type

Directive INSTANTIABLE

- Directives INSTANTIABLE et NOT INSTANTIABLE : capacité d'instancitation d'un type
  Tous les types créés sont par défaut INSTANTIABLE
- NOT INSTANCIABLE : similaire à la notion de classe abstraite
- Chaque type possède
  - un constructeur permettant de créer des objets (persistants ou non) à l'aide de la commande NEW ou au sein d'un commande INSERT
  - un constructeur (par défaut) et plusieurs dans le cas de surcharge
- Un type NOT INSTANCIABLE ne peut pas être FINAL
- Un sous-type NOT INSTANCIABLE peut hériter d'un type INSTANCIABLE

Création d'un type

Directive INSTANTIABLE -- Exemples

```
CREATE TYPE Personnel_t AS OBJECT(
 nom VARCHAR (10), prenom VARCHAR(10), adresse adresse_t))
 NOT INSTANTIABLE NOT FINAL
/

CREATE TYPE Enseignant_t AS OBJECT UNDER Personnel_t(
  Echelon NUMBER, indice NUMBER)
  INSTANTIABLE FINAL
/
```

Types Objet

Suppression d'un type

DROP TYPE nomType [FORCE | VALIDATE] ;

Directives :

- `FORCE` : suppression du type même s'il y a des objets de ce type dans une base
  Oracle marque les colonnes dépendant de ce type, `UNUSED`, et elles deviennent inaccessibles (non recommandé)
- `VALIDATE` : Vérification si les instances du type à supprimer peuvent être substitués par un sur-type.

Exemple :

DROP TYPE Personnel_t FORCE

Types Objet

Création d'un type

Spécification de l'objet

```
CREATE TYPE Bank_Account AS OBJECT (
    acct_number INTEGER(5),
    balance            REAL,
    status             VARCHAR2(10),

    MEMBER PROCEDURE open
          (amount IN REAL),

    MEMBER PROCEDURE verify_acct
          (num IN INTEGER),

    MEMBER PROCEDURE close
          (num IN INTEGER, amount OUT REAL)
);
/

CREATE TYPE BODY Bank_Account AS
    ...
```

Types Objet

Création d'un type

Définition des méthodes associées à l'objet

```
CREATE TYPE BODY Bank_Account AS

    MEMBER PROCEDURE open (amount IN REAL) IS
    BEGIN -- open account with initial deposit
        IF NOT amount > 0 THEN
            RAISE_APPLICATION_ERROR(-20104, 'bad amount');
        END IF;
        -- SELECT acct_sequence.NEXTVAL INTO acct_number FROM dual;
        status := 'open';
        balance := amount;
    END open;
```

Types Objet

Création d'un type

```
    MEMBER PROCEDURE verify_acct (num IN INTEGER) IS
    BEGIN -- check for wrong account number or closed account
        IF (num <> acct_number) THEN
            RAISE_APPLICATION_ERROR(-20105, 'wrong number');
        ELSIF (status = 'closed') THEN
                RAISE_APPLICATION_ERROR(-20106, 'account closed');
        END IF;
    END verify_acct;

    MEMBER PROCEDURE close (num IN INTEGER, amount OUT REAL) IS
    BEGIN -- close account and return balance
        verify_acct(num);
        status := 'closed';
        amount := balance;
    END close;
END;
```

Types Objet

Extraction de la description d'un type

Définition de nouvelles vues du DD pour prendre en compte les types

Exemple :

```
create  type emp_type as object (ninsee varchar2(13),
                                 age number, nom varchar2(30))
/
```

Description de la structure du 1er niveau d'un type :

```
SQL> DESC emp_type
```

Types Objet

Extraction de la description d'un type

Exemples de vues : (`USER_...`, `DBA_...`, `ALL_...`)

Description :

- des collections : `USER_COLL_TYPES`
- des index sur les typesé: `USER_INDEXTYPES`
- des types d'une manière générale : `USER_TYPES`
- des attributs des types : `USER_TYPE_ATTRS`
- des méthodes des types : `USER_TYPE_METHODS`
- des versions des types : `USER_TYPE_VERSIONS`

Passage à l'objet

Tables relationnelles

```
-- Table : MAGASINS2 SQL2
create table MAGASINS2
(   NUMMAG          INTEGER      ,
    NOMMAG          CHAR(30)     ,
    TELMAG          CHAR(15)     ,
    ADRNUMMAG       VARCHAR2(10),
    ADRRUEMAG       VARCHAR2(50),
    ADRCPMAG        VARCHAR2(10),
    ADRVILLEMAG     VARCHAR2(50),
    ADRPAYSMAG      VARCHAR2(50),
    constraint PK_MAGASINS2
        primary key (NUMMAG) );
```

```
-- Table : CLIENTS2 SQL2
create table CLIENTS2
(   NUMCLI          INTEGER      ,
    NOMCLI          CHAR(20)     ,
    TELCLI          CHAR(15)     ,
    ADRNUMCLI       VARCHAR2(10),
    ADRRUECLI       VARCHAR2(50),
    ADRCPCLI        VARCHAR2(10),
    ADRVILLECLI     VARCHAR2(50),
    ADRPAYSCLI      VARCHAR2(50),
    constraint PK_CLIENTS2
        primary key (NUMCLI));
```

```
insert into MAGASINS2 values (1, 'FB', '0145454545', '13', 'Avenue de la paix',
                              '75015', 'Paris', 'France');
```

NUMMAG	NOMMAG	TELMAG	ADRNU	ADRRUEMAG	ADRCP	ADRVILLEMA	ADRPAYSMAG
1	FB	0145454545	13	Avenue de la paix	75015	Paris	France
2	FB	0155555555	20	Avenue de la liberté	06100	Nice	France
3	FB	0155555555	10	Avenue des Amis	6050	Bruxelles	Belgique
4	FB	71226002	10	Avenue du soleil	1001	Tunis	Tunisie

NUMCLI	NOMCLI	TELCLI	ADRNU	ADRRUECLI	ADRCP	ADRVILLECL	ADRPAYSCLI
1	TRAIFOR	0645454545	13	Avenue de la paix	75015	Paris	France
2	CLEMENT	0607080910	17	Avenue de la paix	75015	Paris	France
3	SOUCY	98980307	77	Route de la corniche	4001	Sousse	Tunisie

Tables Objet

Création d'un type – TAD

Première extension du modèle relationnel : Types Abstraits de Données (TAD)
TAD (contexte BD) :

- Nouveau type d'attribut défini par l'utilisateur
  Enrichissement de la collection existante de types disponibles par défaut (`number`, `date`, `char`, `varchar` ...)
- Structure de données partagée
  - Utilisation du type dans une ou plusieurs tables
  - Participation à la composition d'un ou plusieurs autres types

Remarques :

- Un TAD inclut des méthodes qui sont des procédures ou des fonctions
- Elles permettent de manipuler les objets du type abstrait

Tables Objet

Création d'un type – exemple de TAD

```
create type ADRESSE_TYPE as object
  ( ADRNUM     VARCHAR2(10),
    ADRRUE     VARCHAR2(50),
    ADRCP      VARCHAR2(10),
    ADRVILLE   VARCHAR2(50),
    ADRPAYS    VARCHAR2(50) )
/
```

```
create type MAG_TYPE as object
(   NUMMAG   INTEGER ,
    NOMMAG   CHAR(30),
    TELMAG   CHAR(15),
    ADRMAG   ADRESSE_TYPE )
/
```

```
create type CLI_TYPE as object
(   NUMCLI   INTEGER ,
    NOMCLI   CHAR(30),
    TELCLI   CHAR(15),
    ADRCLI   ADRESSE_TYPE )
/
```

Tables Objet

Création d'une table – Exemples

```
create table MAGASINS3 OF MAG_TYPE
( constraint PK_MAGASINS3 primary key (NUMMAG) );

create table CLIENTS3 OF CLI_TYPE
( constraint PK_CLIENTS3  primary key (NUMCLI) );
```

Tables Objet

Création d'un type

Remarques :

- Un type ne peut pas contenir de contraintes (`NOT NULL`, `CHECK`, `UNIQUE`, `DEFAULT`, `PRIMARY KEY`, `FOREIGN KEY`, etc.).
- Les contraintes doivent être déclarées au niveau de la table objet

Accès à la description des types à partir du Dictionnaire de Données :

```
SQL > select table_name, object_id_type, table_type_owner,
      table_type from user_object_tables;
```

Tables Objet

Création/description d'une table Exemples

```
SQL> desc clients2
 Nom                                       NULL ?   Type
 ----------------------------------------- -------- ----------------------------
 NUMCLI                                    NOT NULL NUMBER(38)
 NOMCLI                                             CHAR(20)
 TELCLI                                             CHAR(15)
 ADRNUMCLI                                          VARCHAR2(10)
 ADRRUECLI                                          VARCHAR2(50)
 ADRCPCLI                                           VARCHAR2(10)
 ADRVILLECLI                                        VARCHAR2(50)
 ADRPAYSCLI                                         VARCHAR2(50)

SQL> desc clients3
 Nom                                       NULL ?   Type
 ----------------------------------------- -------- ----------------------------
 NUMCLI                                    NOT NULL NUMBER(38)
 NOMCLI                                             CHAR(30)
 TELCLI                                             CHAR(15)
 ADRCLI                                             ADRESSE_TYPE
```

Tables Objet

Object identifier (OID)

- OID basés sur la clé primaire : Utilisation de l'option *primary key*

  Exemple :

```
create table CLIENTS3 OF CLI_TYPE
   ( constraint PK_CLIENTS3  primary key (NUMCLI) )
   object identifier is primary key ;
```

- Index sur OID :

```
create table CLIENTS3 OF CLI_TYPE
   ( constraint PK_CLIENTS3  primary key (NUMCLI) )
   object identifier is system generated OIDINDEX ndxclients3;

create table CLIENTS3 OF CLI_TYPE
   ( constraint PK_CLIENTS3  primary key (NUMCLI) )
   object identifier is system generated
   OIDINDEX ndxclients3 (storage (initial 100K next 50k
                          minextents 1 maxextents 50)
                        );
```

Tables Objet

Instanciation - exemples

Insertion d'une « ligne » (ou plutôt d'un objet) :

```
insert into MAGASINS3 values (MAG_TYPE(1, 'FB', '0145454545',
          ADRESSE_TYPE('13', 'Avenue de la paix', '75015', 'Paris', 'France')));
insert into MAGASINS3 values (MAG_TYPE(2, 'FB', '0155555555',
          ADRESSE_TYPE('20', 'Avenue de la liberté', '06100', 'Nice', 'France')));
insert into MAGASINS3 values (MAG_TYPE(3, 'FB', '0155555555',
          ADRESSE_TYPE('10', 'Avenue des Amis', '6050', 'Bruxelles', 'Belgique')));
insert into MAGASINS3 values (MAG_TYPE(4, 'FB', '71226002',
          ADRESSE_TYPE('10', 'Avenue du soleil', '1001', 'Tunis', 'Tunisie')));
```

```
SQL> select * from magasins3 ;
NUMMAG   NOMMAG   TELMAG       ADRMAG(ADRNUM, ADRRUE, ADRCP, ADRVILLE, ADRPAYS)
 1         FB     0145454545    ADRESSE_TYPE('13', 'Avenue de la paix',
                                  '75015', 'Paris', 'France')
 2         FB     0155555555    ADRESSE_TYPE('20', 'Avenue de la liberté',
                                  '06100', 'Nice', 'France')
 3         FB     0155555555    ADRESSE_TYPE('10', 'Avenue des Amis',
                                  '6050', 'Bruxelles', 'Belgique')
 4         FB     71226002      ADRESSE_TYPE('10', 'Avenue du soleil',
                                  '1001', 'Tunis', 'Tunisie')
```

Tables Objet

Instanciation - exemples

```
insert into CLIENTS3 values (CLI_TYPE(1, 'TRAIFOR', '0645454545',
              ADRESSE_TYPE('13', 'Avenue de la paix', '75015', 'Paris', 'France')));
insert into CLIENTS3 values (CLI_TYPE(2, 'CLEMENT', '0607080910',
              ADRESSE_TYPE('17', 'Avenue de la paix', '75015', 'Paris', 'France')));
insert into CLIENTS3 values (CLI_TYPE(3, 'SOUCY', '98980307',
              ADRESSE_TYPE('77', 'Route de la corniche', '4001', 'Sousse', 'Tunisie')));
```

```
SQL> Select * from clients3 ;
NUMCLI   NOMCLI      TELCLI     ADRCLI(ADRNUM, ADRRUE, ADRCP, ADRVILLE, ADRPAYS)
  1      TRAIFOR   0645454545   ADRESSE_TYPE('13', 'Avenue de la paix', '75015',
                                            'Paris', 'France')
  2      CLEMENT   0607080910   ADRESSE_TYPE('17', 'Avenue de la paix', '75015',
                                            'Paris', 'France')
  3      SOUCY      98980307    ADRESSE_TYPE('77', 'Route de la corniche', '4001',
                                            'Sousse', 'Tunisie')
```

Tables Objet

Instanciation

Table objet-relationnelle :

- Table dépendante d'un type
- Enregistrements (lignes) dans cette table considérés comme des objets car ils possèdent tous un OID (Object Identifier) unique

```
SQL> SELECT * FROM clients3 ;
NUMCLI   NOMCLI     TELCLI       ADRCLI(ADRNUM, ADRRUE, ADRCP, ADRVILLE, ADRPAYS)
 1       TRAIFOR    0645454545    ADRESSE_TYPE('13', 'Avenue de la paix', '75015',
                                                'Paris', 'France')
 2       CLEMENT    0607080910    ADRESSE_TYPE('17', 'Avenue de la paix', '75015',
                                                'Paris', 'France')
 3       SOUCY       98980307     ADRESSE_TYPE('77', 'Route de la corniche', '4001',
                                                'Sousse', 'Tunisie')
```

Tables Objet

Instanciation

- Renvoi des OID des objets de la table :

```
SQL> SELECT REF(c) FROM clients3 c ;

REF(C)
--------------------------------------------------------------------------------
0000280209E9E229206EDF47DF9996946C4BBD571C4EB9AF259F2F42BC813
00002802095501 41E8898C4859AF0F3D48FA3041944EB9AF259F2F42BC813
0000280209C2C96804847047F6856499690AAC9E254EB9AF259F2F42BC813
```

Tables Objet

Mises à jour

Modifications/Suppressions de «lignes» ou d'objets

- Mise à jour d'une colonne standard

```
update clients3
set NOMCLI = 'CBON' where NUMCLI=2;
```

- Modification d'une colonne appartenant à un type imbriqué

```
update clients3 c
set c.ADRCLI.ADRVILLE = 'MAVILLE' where c.NUMCLI=2;
```

- Suppression d'objet

```
delete from clients3
where numcli = 3 ;

delete from clients3 c
where upper(c.ADRCLI.ADRPAYS) = 'FRANCE' ;
```

Tables Objet

Interrogations

- Utilisation de colonnes standards

```
select numcli, nomcli from clients3;
```

NUMCLI	NOMCLI
1	TRAIFOR
2	CLEMENT
3	SOUCY

- Utilisation d'une colonne appartenant à un type imbriqué

```
select numcli, nomcli,c.ADRCLI.ADRPAYS
from clients3 c;
```

NUMCLI	NOMCLI	ADRCLI.ADRPAYS
1	TRAIFOR	France
2	CLEMENT	France
3	SOUCY	Tunisie

Tables Objet

Interrogations

- avec formatage

```
col nom format A10
col loc format A15
select numcli as cli, nomcli as nom,
  c.ADRCLI.ADRVILLE || ' ' || c.ADRCLI.ADRPAYS as loc
from clients3 c;
```

CLI	NOM	LOC
1	TRAIFOR	Paris France
2	CLEMENT	Paris France
3	SOUCY	Sousse Tunisie

Tables Objet

Interrogations

- avec contraintes

```
SQL> col c.ADRCLI.ADRPAYS format A10
SQL> col c.ADRCLI.ADRVILLE format A10
SQL> select numcli, nomcli,c.ADRCLI.ADRPAYS,
  2  c.ADRCLI.ADRVILLE from clients3 c
  3  WHERE upper(c.ADRCLI.ADRVILLE) like 'P%';
```

NUMCLI	NOMCLI	ADRCLI.ADRPAYS	ADRCLI.ADRVILLE
1	TRAIFOR	France	Paris
2	CLEMENT	France	Paris

Tables imbriquées

(*NESTED TABLE*)

Table imbriquée (`NESTED TABLE`) : collection non ordonnée et non limitée d'éléments de même type

Exemple : table `Departement`

1 table contenant une colonne (table) : Association du type 1-N

NumDep	Budget	Employés		
		NInsee	Nom	Age

??? 1 ou plusieurs tables : Association du type N-N

Tables imbriquées (NESTED TABLE)

Création

```
create  type emp_type as object
        (ninsee varchar2(13), age number, nom varchar2(30))
/

create  type emps_type as table of emp_type
/
create  type departement_type as object
        (numdep varchar2(11), budget number,
         employes emps_type)
/
create table departement of departement_type
        (primary key (numdep))
         nested table employes store as tabemp
/
```

- clause NESTED TABLE : définition d'une table imbriquée
- clause STORE AS : nommage de la structure interne qui stocke les «enregistrements» de cette table imbriquée

Tables imbriquées (NESTED TABLE)

Exemple

```
create  type emp_type as object (ninsee varchar2(13),
                                 age number, nom varchar2(30))
/
create  type emps_type as table of emp_type
/
create  type departement_type as object (numdep varchar2(11),
                                budget number, employes emps_type)
/
create table departement of departement_type
                          (primary key (numdep))
                          nested table employes store as tabemp
/
```

Tables imbriquées (NESTED TABLE)

Exemple

```
SQL> desc departement
 Nom                                       NULL ?   Type
 ----------------------------------------- -------- ------------------------
 NUMDEP                                    NOT NULL VARCHAR2(11)
 BUDGET                                             NUMBER
 EMPLOYES                                           EMPS_TYPE

SQL> desc emps_type
 emps_type TABLE OF EMP_TYPE
 Nom                                       NULL ?   Type
 ----------------------------------------- -------- ------------------------
 NINSEE                                             VARCHAR2(13)
 AGE                                                NUMBER
 NOM                                                VARCHAR2(30)
```

Tables imbriquées (NESTED TABLE)

Insertion

- Insertion des données dans une table imbriquée

```
insert into departement values ('D1', 100000, emps_type());
insert into departement values ('D2', 200000, emps_type());
```

```
SQL> select * from departement ;
NUMDEP               BUDGET EMPLOYES(NINSEE, AGE, NOM)
-------------------- ------ ---------------------------------
D1                   100000 EMPS_TYPE()
D2                   200000 EMPS_TYPE()
```

Tables imbriquées (NESTED TABLE)

Insertion

- Attention : dans l'exemple suivant, la table vide est non initialisée

```
insert into departement (numdep, budget)
            values ('D3', 300000);
```

```
SQL> select * from departement ;
NUMDEP               BUDGET EMPLOYES(NINSEE, AGE, NOM)
-------------------- ------ ------------------------------------
D1                   100000 EMPS_TYPE()
D2                   200000 EMPS_TYPE()
D3                   300000
```

Tables imbriquées (NESTED TABLE)

Insertion

- Insertion des données dans une table imbriquée

```
insert into departement values ('D4', 400000,
                emps_type(emp_type('N5', 25, 'Bibi'),
                          emp_type('N6', 26, 'Cici'),
                          emp_type('N7', 27, 'Didi'),
                          emp_type('N8', 28, 'Fifi')));
```

Tables imbriquées (NESTED TABLE)

Insertion

```
SQL> select * from departement ;
NUMDEP             BUDGET EMPLOYES(NINSEE, AGE, NOM)
---------- ---------- ------------------------------------
D1                 100000 EMPS_TYPE()
D2                 200000 EMPS_TYPE()
D3                 300000
D4                 400000 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                                    EMP_TYPE('N6', 26, 'Cici'),
                                    EMP_TYPE('N7', 27, 'Didi'),
                                    EMP_TYPE('N8', 28, 'Fifi'))
```

Remarque : Commande `INSERT` avec les constructeurs des types de la `NESTED TABLE`

- stocke un objet dans la table
- initialise la table imbriquée associée avec des enregistrements

Tables imbriquées (NESTED TABLE)

Insertion

- Insertion des données dans une table imbriquée

```
insert into departement values ('D5', 400000,
            emps_type(emp_type('N5', 25, 'Bibi'),
                      emp_type('N8', 28, 'Fifi')));
```

```
SQL> select * from departement ;
NUMDEP      BUDGET EMPLOYES(NINSEE, AGE, NOM)
---------- ---------- ----------------------------------------
D1             100000 EMPS_TYPE()
D2             200000 EMPS_TYPE()
D3             300000
D4             400000 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                                EMP_TYPE('N6', 26, 'Cici'),
                                EMP_TYPE('N7', 27, 'Didi'),
                                EMP_TYPE('N8', 28, 'Fifi'))
D5             400000 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                                EMP_TYPE('N8', 28, 'Fifi'))
```

Tables imbriquées (NESTED TABLE)

Insertion avec THE

- Insertion avec THE dans une table imbriquée
  (D1 et D2 étaient initialisés à vides)

```
insert into THE (select d.employes from departement d
                 where d.numdep = 'D1')
        values ('N1', 21, 'CLEMENT');
insert into THE (select d.employes from departement d
                 where d.numdep = 'D2')
        values ('N2', 22, 'CLEMENTINE');
```

Tables imbriquées (NESTED TABLE)

Insertion avec THE

```
SQL> select * from departement ;
NUMDEP      BUDGET EMPLOYES(NINSEE, AGE, NOM)
---------- ---------- ----------------------------------------------------------
D1             100000 EMPS_TYPE(EMP_TYPE('N1', 21, 'CLEMENT'))
D2             200000 EMPS_TYPE(EMP_TYPE('N2', 22, 'CLEMENTINE'))
D3             300000
D4             400000 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                                EMP_TYPE('N6', 26, 'Cici'),
                                EMP_TYPE('N7', 27, 'Didi'),
                                EMP_TYPE('N8', 28, 'Fifi'))
```

Remarques :

- Commande `INSERT INTO THE (SELECT ...)` : stockage d'un enregistrement dans la table imbriquée désignée par `THE`
- `SELECT` après le `THE` : Retourne un seul objet, ce qui permet de sélectionner la table imbriquée associée

Tables imbriquées (NESTED TABLE)

Insertion avec THE

- Insertion avec THE dans une table imbriquée

  (D3 n'était pas initialisé à vide)
  Insertion d'un employé dans le département D3
  alors que celui-ci n'a pas été initialisé

```
insert into THE (select d.employes from departement d
                 where d.numdep = 'D3')
                 values ('N3', 23, 'NE MARCHE PAS');
```

Tables imbriquées (NESTED TABLE)

Insertion avec THE

```
SQL> insert into the (select d.employes from departement d
  2  where d.numdep = 'D3') values ('N3', 23, 'NEMARCHEPAS');
insert into the (select d.employes from departement d
                                    where d.numdep = 'D3')
*
ERREUR à la ligne 1 :
ORA-22908: référence à une valeur de table NULL
```

Explications :

1. Le département D3 est bien un objet de la table Département
2. mais il ne possède pas de table imbriquée
3. car celle-ci n'a pas été créée lors de l'insertion.

Il faut détruire l'objet D3 puis le recréer !

Tables imbriquées (NESTED TABLE)

Modification

- Mise à jour de la table principale

```
update departement d
       set d.budget = d.budget * 1.5
              where d.budget <= 200000 ;
```

```
SQL> select * from departement ;
NUMDEP     BUDGET EMPLOYES(NINSEE, AGE, NOM)
---------- ---------- ----------------------------------------------------------
D1             150000 EMPS_TYPE(EMP_TYPE('N1', 21, 'CLEMENT'))
D2             300000 EMPS_TYPE(EMP_TYPE('N2', 22, 'CLEMENTINE'))
D3             300000
D4             400000 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                                EMP_TYPE('N6', 26, 'Cici'),
                                EMP_TYPE('N7', 27, 'Didi'),
                                EMP_TYPE('N8', 28, 'Fifi'))
D5             400000 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                                EMP_TYPE('N8', 28, 'Fifi'))
```

Tables imbriquées (NESTED TABLE)

Modification (suite)

- Mise à jour de la table principale selon un prédicat dans la table imbriquée

```
update departement d set d.budget = d.budget + 777
        where  exists (select * from
        the ( select dt.employes from departement dt
              where dt.numdep =
              d.numdep ) nt
      where nt.age < 25 ) ;
```

Description :

- Requête qui retourne les employés de chaque département

```
select dt.employes from departement dt
                  where dt.numdep = d.numdep
```

- Condition sur un attribut de la table imbriquée :

```
where nt.age < 25
```

- Alias de la table imbriquée : `nt`

Tables imbriquées (NESTED TABLE)

Modification (suite)

- Mise à jour de la table principale selon un prédicat dans la table imbriquée

Tables imbriquées (NESTED TABLE)

Modification (suite)

```
SQL> select * from departement ;
NUMDEP   BUDGET EMPLOYES(NINSEE, AGE, NOM)
------ -------- ----------------------------------------------------------
D1       150777 EMPS_TYPE(EMP_TYPE('N1', 21, 'CLEMENT'))
D2       300777 EMPS_TYPE(EMP_TYPE('N2', 22, 'CLEMENTINE'))
D3       300000
D4       400000 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N6', 26, 'Cici'),
                          EMP_TYPE('N7', 27, 'Didi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
D5       400000 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
```

Tables imbriquées (NESTED TABLE)

Modification

- Mise à jour de la table principale selon un prédicat dans la table imbriquée

```
update departement d set d.budget = d.budget + 999
        where  exists
              (select * from the
              (select dt.employes from departement dt
                            where dt.numdep = d.numdep ) nt
              where nt.age > 25 ) ;
```

Tables imbriquées (NESTED TABLE)

Modification

```
SQL> select * from departement ;
NUMDEP   BUDGET EMPLOYES(NINSEE, AGE, NOM)
------ -------- ----------------------------------------
D1       150777 EMPS_TYPE(EMP_TYPE('N1', 21, 'CLEMENT'))
D2       300777 EMPS_TYPE(EMP_TYPE('N2', 22, 'CLEMENTINE'))
D3       300000
D4       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N6', 26, 'Cici'),
                          EMP_TYPE('N7', 27, 'Didi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
D5       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
```

Remarque : les mêmes employés sont dans deux départements

Tables imbriquées (NESTED TABLE)

Modification

- Mise à jour dans la table imbriquée

```
update
    the (select d.employes from departement d
                            where d.numdep = 'D2' ) nt
    set nt.age = 44
        where nt.ninsee = 'N2' ;
```

Tables imbriquées (NESTED TABLE)

Modification

```
SQL> select * from departement ;
NUMDEP   BUDGET EMPLOYES(NINSEE, AGE, NOM)
------- ------- ----------------------------------------------------
D1       150777 EMPS_TYPE(EMP_TYPE('N1', 21, 'CLEMENT'))
D2       300777 EMPS_TYPE(EMP_TYPE('N2', 44, 'CLEMENTINE'))
D3       300000
D4       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N6', 26, 'Cici'),
                          EMP_TYPE('N7', 27, 'Didi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
D5       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
```

Remarque : Il est impossible de modifier plusieurs enregistrements de différentes tables imbriquées avec une seule commande UPDATE !

Tables imbriquées (NESTED TABLE)

Suppression

- Suppression dans la table principale

```
delete from departement
       where numdep = 'D3' ;
```

```
SQL> select * from departement ;
NUMDEP   BUDGET EMPLOYES(NINSEE, AGE, NOM)
_______ _______ ____________________________________________________________
D1       150777 EMPS_TYPE(EMP_TYPE('N1', 21, 'CLEMENT'))
D2       300777 EMPS_TYPE(EMP_TYPE('N2', 44, 'CLEMENTINE'))
D4       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N6', 26, 'Cici'),
                          EMP_TYPE('N7', 27, 'Didi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
D5       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
```

Tables imbriquées (NESTED TABLE)

Suppression

- Suppression à partir d'une valeur de la table imbriquée
  Elimination des départements qui emploient une personne dont le nom est `fifi`

```
delete from departement d
        where exists (select * from
        the (select dt.employes from departement dt
                          where dt.numdep = d.numdep) nt
        where upper(nt.nom) like '%FIFI%' );
```

Tables imbriquées (NESTED TABLE)

Suppression

```
SQL> select * from departement ;
NUMDEP    BUDGET EMPLOYES(NINSEE, AGE, NOM)
------- -------- ----------------------------------------------------------
D1         150777 EMPS_TYPE(EMP_TYPE('N1', 21, 'CLEMENT'))
D2         300777 EMPS_TYPE(EMP_TYPE('N2', 44, 'CLEMENTINE'))
```

Tables imbriquées (NESTED TABLE)

Suppression

- Suppression d'une table imbriquée
  Elimination des départements qui emploient une personne dont le nom est `fifi`

```
delete the (select dt.employes
            from departement dt where dt.numdep = 'D1') nt
            where nt.nom = 'CLEMENT';
```

```
SQL> select * from departement ;
NUMDEP    BUDGET EMPLOYES(NINSEE, AGE, NOM)
------- -------- ----------------------------------------------------------
D1         150777
D2         300777 EMPS_TYPE(EMP_TYPE('N2', 44, 'CLEMENTINE'))
```

Ingénieurs sup Galilée

Tables imbriquées (NESTED TABLE)

Interrogation

- *Quels sont les numéros et les noms des employés du département D4 ?*

```
SQL> select * from departement ;
NUMDEP   BUDGET EMPLOYES(NINSEE, AGE, NOM)
------- ------- ----------------------------------------------------------
D1       150777 EMPS_TYPE(EMP_TYPE('N1', 21, 'CLEMENT'))
D2       300777 EMPS_TYPE(EMP_TYPE('N2', 44, 'CLEMENTINE'))
D4       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N6', 26, 'Cici'),
                          EMP_TYPE('N7', 27, 'Didi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
D5       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
```

Tables imbriquées (NESTED TABLE)

Interrogation

```
select nt.ninsee, nt.nom
       from the (select dt.employes from departement dt
       where dt.numdep = 'D4') nt;
```

NINSEE	NOM
N5	Bibi
N6	Cici
N7	Didi
N8	Fifi

Tables imbriquées (NESTED TABLE)

Interrogation

- *Quels sont les numéros et les noms des employés du département D4 qui ont moins de 26 ans ?*

```
SQL> select * from departement ;
NUMDEP   BUDGET EMPLOYES(NINSEE, AGE, NOM)
------- ------- ----------------------------------------------------------
D1       150777 EMPS_TYPE(EMP_TYPE('N1', 21, 'CLEMENT'))
D2       300777 EMPS_TYPE(EMP_TYPE('N2', 44, 'CLEMENTINE'))
D4       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N6', 26, 'Cici'),
                          EMP_TYPE('N7', 27, 'Didi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
D5       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
```

Tables imbriquées (NESTED TABLE)

Interrogation

```
select nt.ninsee, nt.nom
       from the (select dt.employes from departement dt
       where dt.numdep = 'D4') nt where nt.age < 26;
```

NINSEE	NOM
N5	Bibi

Tables imbriquées (NESTED TABLE)

Interrogation

- Requête : Quel est le nombre d'employés du département D4 ?

```
SQL> select * from departement ;
NUMDEP   BUDGET EMPLOYES(NINSEE, AGE, NOM)
------- ------- ----------------------------------------------------------
D1       150777 EMPS_TYPE(EMP_TYPE('N1', 21, 'CLEMENT'))
D2       300777 EMPS_TYPE(EMP_TYPE('N2', 44, 'CLEMENTINE'))
D4       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N6', 26, 'Cici'),
                          EMP_TYPE('N7', 27, 'Didi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
D5       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
```

Tables imbriquées (NESTED TABLE)

Interrogation

```
select COUNT(*) "Nombre d'employés"
        from the (select dt.employes from departement dt
        where dt.numdep = 'D4') nt ;
Nombre d'employés
------------------
                 4
```

Ingénieurs sup Galilée

Tables imbriquées (NESTED TABLE)

Interrogation

- *Quels sont les numéros et les noms des employés des départements D1 et D2 ?*

```
SQL> select * from departement ;
NUMDEP   BUDGET EMPLOYES(NINSEE, AGE, NOM)
------- ------- ----------------------------------------------------------
D1       150777 EMPS_TYPE(EMP_TYPE('N1', 21, 'CLEMENT'))
D2       300777 EMPS_TYPE(EMP_TYPE('N2', 44, 'CLEMENTINE'))
D4       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N6', 26, 'Cici'),
                          EMP_TYPE('N7', 27, 'Didi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
D5       400999 EMPS_TYPE(EMP_TYPE('N5', 25, 'Bibi'),
                          EMP_TYPE('N8', 28, 'Fifi'))
```

Tables imbriquées (NESTED TABLE)

Interrogation

```
select select nt.ninsee, nt.nom from the
       (select dt.employes from departement dt
                    where dt.numdep = 'D1') nt
union
select nt.ninsee, nt.nom from the
       (select dt.employes from departement dt
                    where dt.numdep = 'D2') nt;
```

NINSEE	NOM
N1	CLEMENT
N2	CLEMENTINE

Plusieurs Tables imbriquées

Création

Regroupement des tables imbriquées Professeurs et Formations dans la table Cours

NumC	Titre	Professeurs		Formations	
		Nom	Spécialité	Filière	Horaire

```
create type prof_type as object
       (nom varchar2(30), specialite varchar2(30))
/
create type profs_type as table of prof_type
/
create type formation_type as object
       (filiere varchar2(30), horaire number(5))
/
create type formations_type as table of formation_type
/
```

Plusieurs Tables imbriquées

Création

```
create  type prof_type as object
        (nom varchar2(30), specialite varchar2(30))
/
create  type profs_type as table of prof_type
/
create  type formation_type as object
        (filiere varchar2(30), horaire number(5))
/
create  type formations_type as table of formation_type
/
create  type cours_type as object
        (numc varchar2(5), titre varchar2(15),
         professeurs profs_type, formations formations_type)
/
create table cours of cours_type
 (constraint pk_cours primary key (numc))
  nested table professeurs store as tabprofs,
  nested table formations store as tabformations ;
```

Plusieurs Tables imbriquées

Insertion

Insertion d'un objet dans la table Cours, sans le lier à des professeurs ou à des formations

```
insert into cours values ('BD', 'Bases de Données',
                          profs_type(), formations_type());
```

```
select * from cours;
```

NUMC	TITRE	PROFESSEURS(NOM, SPECIALITE)	FORMATIONS(FILIERE, HORAIRE)
BD	Bases de Données	PROFS_TYPE()	FORMATIONS_TYPE()

Plusieurs Tables imbriquées

Insertion

- Insertion, avec `VALUES`, dans 2 tables imbriquées :

```
insert into cours values ('DW', 'Data WareHouse',

profs_type(prof_type('Clémence','BD'),
           prof_type('Adam', 'BD')),

formations_type(formation_type('Master1', 100),
                formation_type('DESS EID', 200),
                formation_type('DEA AIOC', 200) ));
```

Plusieurs Tables imbriquées

Insertion

```
select * from cours;
```

```
NUMC TITRE                PROFESSEURS(NOM, SPECIALITE)                FORMATIONS(FILIERE, HORAI
---- -------------------- ------------------------------------------- -------------------------
BD   Bases de Données     PROFS_TYPE()                                FORMATIONS_TYPE()
DW   Data WareHouse       PROFS_TYPE(                                 FORMATIONS_TYPE(
                            PROF_TYPE('Clémence', 'BD'),                FORMATION_TYPE('Master
                            PROF_TYPE('Adam', 'BD'))                    FORMATION_TYPE('Master
                                                                        FORMATION_TYPE('Master
```

Plusieurs Tables imbriquées

Insertion

Insertion, avec VALUES, dans 2 tables imbriquées

```
insert into cours values ('BDA',
                  'Bases de Données Avancées',

profs_type(prof_type('Clémence','BD'),
      prof_type('Traifor', 'BD'),
      prof_type('Le Bon', 'BD')),

formations_type(
 formation_type('Master 2P', 200),
 formation_type('Master 2R', 200) ));
```

Plusieurs Tables imbriquées

Insertion

Cours

NumC	Titre	Professeurs		Formations	
		Nom	Spécialité	Filière	Horaire
BD	Bases de données				
DW	DataWareHouse	Clémence	BD	Master 1	100
		Adam	BD	Master 2P	200
				Master 2R	200
BDA	Bases de données avancés	Clémence	BD	Master 2P	
		Traifor	BD	Master 2R	
		Le Bon	BD		

L'affichage SQL+ est très mauvais ...

```
select * from cours;

NUMC  TITRE                PROFESSEURS(NOM, SPECIALITE)  FORMATIONS(FILIERE, HORAIRE)
BD    Bases de Données     PROFS_TYPE()                  FORMATIONS_TYPE()
DW    Data WareHouse       PROFS_TYPE(PROF_TYPE('Clémence', 'BD'), PROF_TYPE('Adam', 'BD'))
                                                         FORMATIONS_TYPE(FORMATION_TYPE('M
FORMATION_TYPE('Master 2P', 200), FORMATION_TYPE('Master 2R', 200))
BDA   Bases de Données Avancées PROFS_TYPE(PROF_TYPE('Clémence', 'BD'), PROF_TYPE('Traif
PROF_TYPE('Le Bon', 'BD'))
                                                         FORMATIONS_TYPE(FORMATION_TYPE('M

FORMATION_TYPE('Master 2R', 200))
```

Plusieurs Tables imbriquées

Insertion

Insertion, avec THE et VALUES, dans 2 tables imbriquées

Enregistrement des données : les professeurs Traifor et Parisi enseignent les BD

```
insert into the (select c.professeurs from cours c
                 where numc = 'BD') values ('Traifor', 'SI') ;
insert into the (select c.professeurs from cours c
                 where numc = 'BD') values ('Parisi', 'DM');

select * from cours;
```

NumC	Titre	Professeurs		Formations	
		Nom	Spécialité	Filière	Horaire
BD	Bases de données	Traifor	SI		
		Parisi	DM		
DW	DataWareHouse	Clémence	BD	Master 1	100
		Adam	BD	Master 2P	200
				Master 2R	200
BDA	Bases de données avancés	Clémence	BD	Master 2P	
		Traifor	BD	Master 2R	
		Le Bon	BD		

Plusieurs Tables imbriquées

Insertion

Insertion, avec THE et VALUES, dans 2 tables imbriquées

Le cours BD :

- appartient au cursus INFO1
- requiert un volume horaire de 70 heures

```
insert into the (select c.formations from cours c
                 where numc = 'BD') values ('INFO1', 70);

select * from cours;
```

NumC	Titre	Professeurs		Formations	
		Nom	Spécialité	Filière	Horaire
BD	Bases de données	Traifor	SI	INFO1	70
		Parisi	DM		
DW	DataWareHouse	Clémence	BD	Master 1	100
		Adam	BD	Master 2P	200
				Master 2R	200
BDA	Bases de données avancés	Clémence	BD	Master 2P	
		Traifor	BD	Master 2R	
		Le Bon	BD		

Plusieurs Tables imbriquées

Insertion

Insertion , avec THE et SELECT, dans 2 tables imbriquées

Le cours BD doit être enseigné désormais dans toutes les filières concernées par la matière DW à condition que celle-ci aient un volume de moins de 150 heures

```
insert into the (select c.formations from cours c
                 where c.numc = 'BD')
        select nestedf.filiere, nestedf.horaire
               from the (select c.formations from cours c
                         where c.numc = 'DW') nestedf
        where nestedf.horaire < 150;
```

Plusieurs Tables imbriquées

Insertion

```
select * from cours;
```

NumC	Titre	Professeurs		Formations	
		Nom	Spécialité	Filière	Horaire
BD	Bases de données	Traifor	SI	INFO1	70
		Parisi	DM	Master 1	
DW	DataWareHouse	Clémence	BD	Master 1	100
		Adam	BD	Master 2P	200
				Master 2R	200
BDA	Bases de données avancés	Clémence	BD	Master 2P	
		Traifor	BD	Master 2R	
		Le Bon	BD		

Plusieurs Tables imbriquées

Modification – Exemple

Dans la matière Data WareHouse, le professeur Adam est remplacé par le professeur Saitout et que les horaires pour le Master 2P augmentent de 30%

```
update the (select c.professeurs from cours c
            where c.titre ='Data WareHouse') nestedprf
set nestedprf.nom = 'Saitou' where nestedprf.nom = 'Adam';

update the (select c.formations from cours c
            where c.titre ='Data WareHouse') nestedfrm
set nestedfrm.horaire = horaire * 1.3
            where nestedfrm.filiere like 'Master 2P%';
```

Plusieurs Tables imbriquées

Modification

NumC	Titre	Professeurs		Formations	
		Nom	Spécialité	Filière	Horaire
BD	Bases de données	Traifor	SI	INFO1	70
		Parisi	DM	Master 1	
DW	DataWareHouse	Clémence	BD	Master 1	100
		Saitou	BD	Master 2P	**290**
				Master 2R	200
BDA	Bases de données avancés	Clémence	BD	Master 2P	
		Traifor	BD	Master 2R	
		Le Bon	BD		

Plusieurs Tables imbriquées

Modification – Exemple

Explications :

- Modification à l'aide de la commande `UPDATE` d'un ou plusieurs attributs dans une des deux tables imbriquées de la table cours
- Modification d'un professeurs et une formation dans le cadre d'une matière donnée : 2 requêtes UPDATE distinctes (car les 2 tables imbriquées sont concernées)
- Nécessite de recourir à un alias pour identifier l'objet dans la table imbriquée

Plusieurs Tables imbriquées

Modification

Pour la matière DW, remplacement de la filière Master1 par la filière MASTER 2 et enregistrement d'un volume horaire de 150 heures

```
update the (select c.formations from cours c
            where c.titre = 'Data WareHouse') nestedfrm

set nestedfrm.horaire = 150
            where nestedfrm.filiere = 'Master1';

update the (select c.formations from cours c
                    where c.titre = 'Data WareHouse') nestedfrm
set nestedfrm.filiere = 'MASTER 2'
                    where nestedfrm.filiere = 'Master1';
```

Plusieurs Tables imbriquées

Modification

NumC	Titre	Professeurs		Formations	
		Nom	Spécialité	Filière	Horaire
BD	Bases de données	Traifor	SI	INFO1	70
		Parisi	DM	Master 1	
DW	DataWareHouse	Clémence	BD	**MASTER 2**	150
		Saitou	BD	Master 2P	290
				Master 2R	200
BDA	Bases de données avancés	Clémence	BD	Master 2P	
		Traifor	BD	Master 2R	
		Le Bon	BD		

Plusieurs Tables imbriquées

Suppression

Suppression dans 2 tables imbriquées

Le professeur Parisi n'enseigne plus la matière BD. Enregistrement de cette information

```
delete the (select c.professeurs
            from cours c where c.numc = 'BD') nt
            where nt.nom = 'Parisi';
```

NumC	Titre	Professeurs		Formations	
		Nom	Spécialité	Filière	Horaire
BD	Bases de données	Traifor	SI	INFO1	70
				Master 1	
DW	DataWareHouse	Clémence	BD	MASTER 2	100
		Saitou	BD	Master 2P	290
				Master 2R	200
BDA	Bases de données avancés	Clémence	BD	Master 2P	
		Traifor	BD	Master 2R	
		Le Bon	BD		

Plusieurs Tables imbriquées

Suppression

Suppression dans 2 tables imbriquées

La filière Master1 n'inclut plus la matière BD dans son cursus.

Enregistrement de cette information

```
delete the (select c.formations
            from cours c where c.numc = 'BD') nt
            where nt.filiere = 'Master1';
```

NumC	Titre	Professeurs		Formations	
		Nom	Spécialité	Filière	Horaire
BD	Bases de données	Traifor	SI	INFO1	70
DW	DataWareHouse	Clémence	BD	MASTER 2	100
		Saitou	BD	Master 2P	290
				Master 2R	200
BDA	Bases de données avancés	Clémence	BD	Master 2P	
		Traifor	BD	Master 2R	
		Le Bon	BD		

Plusieurs niveaux d'imbrication

NumC	Titre	Professeurs		Formations		
		Nom	Spécialité	Filière	Horaire	Dates
						Jour
BD	Bases de données	Traifor	SI	INFO1	70	
DW	DataWareHouse	Clémence	BD	MASTER 2	100	
		Saitou	BD	Master 2P	290	
				Master 2R	200	
BDA	Bases de données avancés	Clémence	BD	Master 2P		
		Traifor	BD	Master 2R		
		Le Bon	BD			

Oracle 8 ne permet pas d'implanter plusieurs niveaux d'imbrication dans une table objet-relationnelle

?? dans oracle 9i et/ou 10g ??

Tableaux pré-dimensionnés (VARRAY)

- VARRAY (*Varrying ARRAY*) : collection ordonnée et limitée d'éléments de même type
- Si le nombre d'éléments maximum contenus dans une table imbriquées est connu *a priori*
  possibilité d'utiliser un tableau de type VARRAY à la place d'une table imbriquée
- Exemple : stockage de 3 numéros de téléphone maximum par professeur

Professeurs :

NumP	NomP	Adresse				Téléphones
		AdrNum	AdrRue	AdrVille	AdrCP	NumTel

Tableaux de taille pré-dimensionnée (VARRAY)

Exemple

Stockage de 3 numéros de téléphone maximum par professeur

- Création

```
create  type AAdresse_type as object
        (AdrNum varchar2(10), AdrNom varchar2(30),
         AdrVille varchar2(20), AdrCP varchar2(5))
/
create  type tel_type as object (NumTel varchar2(20))
/

create  type tels_type as varray(3) of tel_type
/

create  type professeur_type as object
(nump varchar2(5), nomp varchar2(20),
          Adresse AAdresse_type, Telephones tels_type)
/
create table professeurs of professeur_type
        (constraint pk_professeurs primary key (nump));
```

Tableaux de taille pré-dimensionnée (VARRAY)

- Insertion : INSERT avec VALUES
  Stockage de 3 objets de type Professeur avec respectivement aucun, trois et deux numéros de téléphone (enregistrements du VARRAY)

```
insert into professeurs values ('P1', 'Clémence',
        AAdresse_type(7, 'Avenue de la Paix', 'Paris', '75009'),
        tels_type());

insert into professeurs values ('P2', 'Adam',
        AAdresse_type(77, 'Rue de la liberté', 'Paris', '75015'),
        tels_type(tel_type('01 53 80 07 99'),
                  tel_type('06 14 56 07 06'),
                  tel_type('01 49 40 07 40')));

insert into professeurs values ('P3', 'Saitou',
        AAdresse_type(1, 'Rue de la liberté', 'Paris', '75015'),
        tels_type(tel_type('01 53 80 53 80'),
        tel_type('06 14 56 14 77'), NULL));
```

Tableaux de taille pré-dimensionnée (VARRAY)

NumP	NomP	Adresse				Téléphones
		AdrNum	AdrRue	AdrVille	AdrCP	NumTel
P1	Clémence	77	Avenue de la paix	Paris	75009	NULL
						NULL
						NULL
P2	Adam	7	Rue de la liberté	Paris	75015	01 53 80 07 99
						06 14 56 07 06
						01 49 40 07 40
P3	Saitou	1	Rue de la liberté	Paris	75015	01 53 80 53 80
						06 14 56 14 77
						NULL

Tableaux de taille pré-dimensionnée (VARRAY)

- Insertion : INSERT dans un VARRAY avec PL/SQL
  - Avec les tableaux VARRAY, l'opérateur THE n'est pas opérationnel (Version 8 d'oracle (à vérifier sur les V9 et V10g)
  - Pour manipuler les tableaux, il est nécessaire d'utiliser un programme PL/SQL

```
DECLARE
  new_tels tels_type :=tels_type(tel_type('01 55 55 55 55'),
                                 tel_type('06 06 98 98 98'),
                                 tel_type('01 40 40 40 40'));
BEGIN
    update professeurs
    set telephones = new_tels
    where nump = 'P1';
END;
/
```

Tableaux de taille pré-dimensionnée (VARRAY)

NumP	NomP	Adresse				Téléphones
		AdrNum	AdrRue	AdrVille	AdrCP	NumTel
P1	Clémence	77	Avenue de la paix	Paris	75009	01 55 55 55 55
						06 06 98 98 98
						01 40 40 40 40
P2	Adam	7	Rue de la liberté	Paris	75015	01 53 80 07 99
						06 14 56 07 06
						01 49 40 07 40
P3	Saitou	1	Rue de la liberté	Paris	75015	01 53 80 53 80
						06 14 56 14 77
						NULL

Remarque :

- Insérer un seul numéro de téléphone pour le professeur P1 et le placer au 2ème rang dans le tableau telephones
  Rédaction ci-dessous de l'instruction d'affectation :

```
new_tels tels_type := tels_type(NULL,
                tel_type('06 06 98 98 98'), NULL);
```

Ingénieurs sup Galilée

Conclusion

Comparaison entre NESTED TABLE et VARRAY

- A vérifier selon les versions d'Oracle
- Possibilité de définir un index dans un `NESTED TABLE`
  Le nombre d'éléments n'est pas limité dans une table imbriquée
- Pas de possibilité de définir d'index dans un `VARRAY`
  Le nombre d'éléments est limité dans une tableau pré-dimensionné
- Possibilité d'accéder directement aux enregistrements stockés dans les deux 2 structures de données
  fonctions : EXISTS, FIRST, LAST, etc.
- Performances ? : *NestedTable* > *Varray*