U.B.S. – I.U.T. de Vannes Licence professionnelle CSD/S2IMA

TP Oracle Objet-relationnel / PostgreSQL Objet-relationnel

On étudie le cas AEROCLUB du contexte OLTP dans un contexte objet-relationnel. La modélisation objet-relationnelle vous est donnée. L'accent est mis sur l'héritage et l'absence de jointure. Les imbrications proposées reposent sur l'hypothèse d'un choix immuable des traitements à privilégier. On vous demande de l'implanter dans Oracle (héritage de types) voire dans PostgreSQL (héritage de tables).

Partie 1 : Modélisation Objet-relationnelle (traitée complètement)

Du modèle NF² au Graphe des types complet

Construction des relations en NF² :

Le schéma relationnel de l'annexe 2 peut s'exprimer en NF² comme ceci :

TYPE_ENGIN{[TypeAvion, NBPlaces, Puissance, PrixHeureMoteur]}

AVION{[NumAvion, TypeAvion, AnneeAvion, Hangar, DateMiseEnService]}

ADHERENT{[NumAdherent, InstructeurSuiveur, NomAdherent, PrenomAdherent, AdresseAdherent]}

INSTRUCTEUR{[NumInstructeur,NumAdherent, SalaireFixe]}

VOL{[NumVol, Directeur, Payeur, Avion, DateVol, DureeVol]}

AUTONOMIES{[NumAdherent, TypeAvion]}

QUALIFICATIONS{[NumInstructeur, TypeAvion]}

Remarque : l'héritage implique une dépendance d'inclusion entre la clé de ADHERENT et sa valeur dans INSTRUCTEUR, donc une clé étrangère NumAdherent qui est clé candidate de la relation INSTRUCTEUR. Elle sera notée en italique pour rappeler qu'il s'agit d'une clé candidate.

Les seules associations N :M qui peuvent être imbriquées sont AUTONOMIES et QUALIFICATIONS qui, de plus, n'expriment pas de dépendances fonctionnelles élémentaires et directes entre attributs. Compte tenu des traitements à privilégier d'affectation des avions et des directeurs de vols à un vol demandé par un adhérent, on décide de la double imbrication dans ADHERENTS de AUTONOMIES et dans INSTRUCTEUR de QUALIFICATIONS.

Le modèle NF² devient donc :

TYPE_ENGIN**{[**TypeAvion, NBPlaces, Puissance, PrixHeureMoteur**]}**

AVION**{[**NumAvion, TypeAvion, AnneeAvion, Hangar, DateMiseEnService**]}**

ADHERENT**{[**NumAdherent, InstructeurSuiveur, NomAdherent, PrenomAdherent, AdresseAdherent, AUTONOMIES**{[**TypeAvion**]}** **]}**

INSTRUCTEUR{[NumInstructeur, *NumAdherent*, SalaireFixe, QUALIFICATIONS{[TypeAvion]}]}

VOL{[NumVol, Directeur, Payeur, Avion, DateVol,DureeVol]}

De plus, on a les dépendances d'inclusion suivantes :

- AVION.TypeAvion inclus dans TYPE_ENGIN.TypeAvion ;
- ADHERENT.InstructeurSuiveur inclus dans INSTRUCTEUR.NumInstructeur ;
- INSTRUCTEUR.NumAdherent inclus dans ADHERENT.NumAdherent ;
- VOL.Directeur inclus dans INSTRUCTEUR.NumInstructeur ;
- VOL.Payeur inclus dans ADHERENT.NumAdherent ;
- VOL.AVION inclus dans AVION.NumAvion ;
- AUTONOMIES.TypeAvion inclus dans TYPE_ENGIN.TypeAvion ;
- QUALIFICATIONS.TypeAvion inclus dans TYPE_ENGIN.TypeAvion.

Lorsque l'ensemble imbriqué contient un seul attribut, les [] sont facultatifs, ce qui est le cas pour AUTONOMIES et QUALIFICATIONS :

ADHERENT{[NumAdherent, InstructeurSuiveur, NomAdherent, PrenomAdherent, AdresseAdherent, AUTONOMIES{TypeAvion}]}

INSTRUCTEUR{[NumInstructeur, *NumAdherent*, SalaireFixe, QUALIFICATIONS{TypeAvion}]}

Graphe des types simple littéral

De ce modèle NF^2, on peut déduire de manière littérale le graphe des types simple associé. On ne tient pas compte des accolades (constructeur d'ensembles) de premier niveau qui traduisent la construction de l'extension vue comme un ensemble d'instances. Chaque utilisation des [] se matérialise sur le schéma par une X et chaque utilisation des {} se matérialise par une *. Les dépendances d'inclusion sont traduites par des liens de référence (matérialisés par des flèches). La seule transformation concerne les noms des types qui suivent le nom de la relation NF^2 : ADHERENT devient ADHERENT_TY, INSTRUCTEUR devient INSTRUCTEUR_TY, etc.

Or en contexte Oracle et PostgreSQL, l'utilisation de * et X doit se faire de manière alternative : ils ne doivent pas se suivre sans être séparés par un nom de constructeur (type intermédiaire) représentant le type de l'élément de la collection.

A noter que si la séquence * * n'est pas possible, X X est toujours possible.

Oracle nécessite que l'on déclare un type pour une collection alors que la déclaration des types tableaux des types utilisateurs déclarés implicitement ou explicitement est automatique dans PostgreSQL. Un tableau est l'unique collection dans PostgreSQL alors que les VARRAY (ordre, limite fixée) et les Nested Tables (opérateurs exclusifs de comparaison, index, pas d'hypothèses, stockage séparé) sont les collections possibles en Oracle.

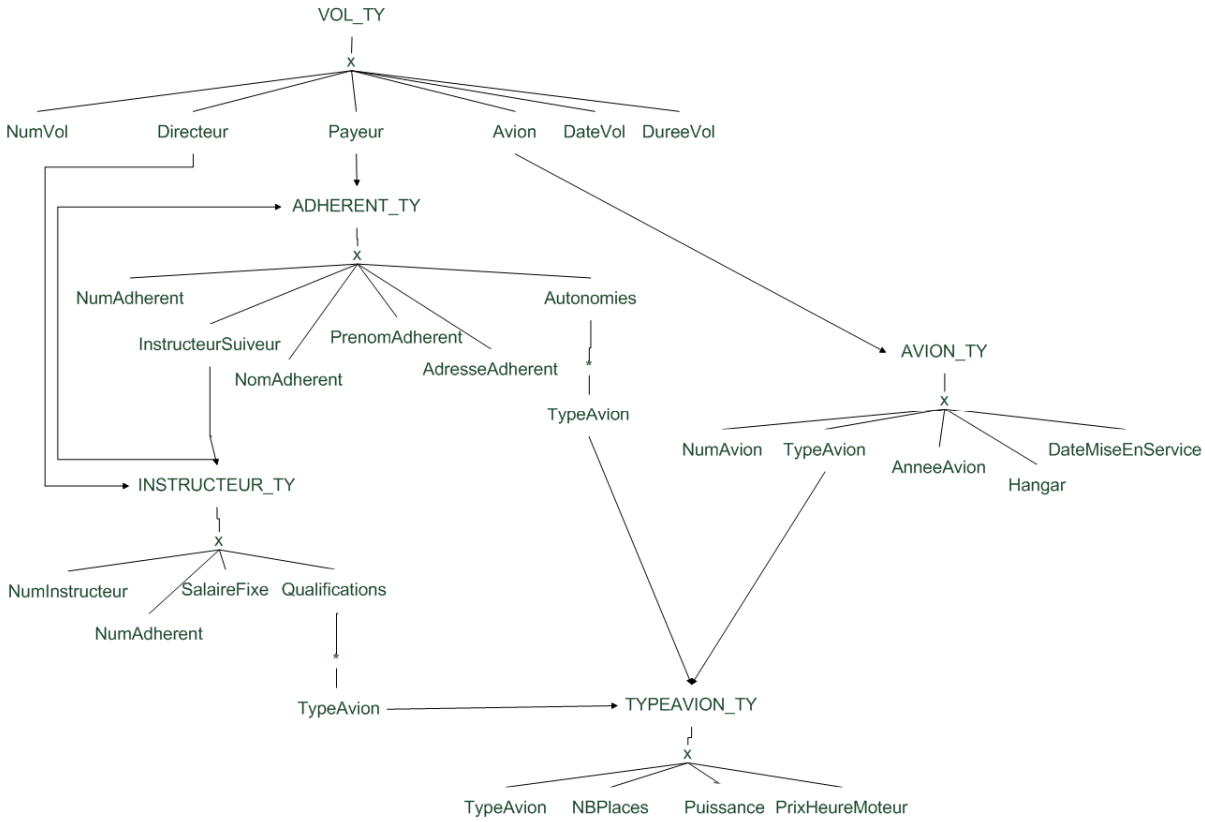


Figure 1 Graphe des Types Simple associé à la modélisation NFNF

Vers le Graphe des Types complet

La modélisation en Graphe de Types Simple, adaptée aux contraintes du contexte Oracle puis à celles du contexte PostgreSQL fait apparaître explicitement :

- Les noms des types sur lesquels seront construites les tables,
- L'association des types simples ou prédéfinis aux attributs.

Ainsi, pour passer de notre modélisation initiale au Graphe de Types Simple, il faut effectuer les transformations suivantes :

- [TR1] lors d'une utilisation successive des opérateurs `*` et `X` qui est rendue obligatoire, on a rajouté les noms de types intermédiaires : `REF_TYPEAVION_TY` utilisé comme élément des collections `AUTONOMIES` et `QUALIFICATIONS`.
- [TR2] les noms des attributs assimilés à des références commencent par `REF`. C'est le cas de `REFInstructeurSuiveur`, `REFDirecteur`, `REFPayeur`, `REFAvion`,
- [TR3] à chaque attribut est associé un type simple (`n` pour numérique, `c` pour chaîne, `b` pour binaire), prédéfini (`d` pour date), référencé (`REFInstructeurSuiveur`, `REFDirecteur`, `REFPayeur`, `REFAvion`), ou construit (constructeur d'ensemble pour Oracle : `NT_REF_TYPEAVION_TY`, `REF_TYPEAVION_TY[]` pour PostgreSQL, constructeur de tuple : `REF_TYPEAVION_TY`). A noter que l'utilisation du type booleen (vrai/faux) étant problématique dans l'implantation du SQL d'Oracle (ce n'est pas le cas pour PostgreSQL), on préfère le type n.

Les collections AUTONOMIES et QUALIFICATIONS ont pour élément des données de type REF_TYPEAVION_TY qui ne contient que TypeAvion et une référence sur TYPEAVION_TY. Ainsi, on peut utiliser ce type déjà existant dans le type AVION_TY au niveau de l'attribut TypeAERO. S'il avait eu d'autres données dans les éléments des collections, on aurait plutôt créé à la place dans TYPEAVION_TY des attributs IDTypeAvion et REFTypeAvion pour ne pas avoir des attributs jamais valués.

La prise en compte de l'héritage implique la factorisation au niveau du sur-type des attributs communs entre le sur-type et tous les sous-types. Ainsi NumAdherent dans INSTRUCTEUR_TY disparaît du graphe d'héritage et donc du graphe complet. Il peut arriver qu'il existe des attributs en communs entre plusieurs sous-types qui ne sont pas présents dans le surtype.

Le bi-référencement permet d'illustrer le concept de lien par valeur et de lien par pointeur (ou référence). Dans le graphe complet, on a dédoublé les liens (bi-référencement) afin d'avoir des liens par valeur couplés avec les liens par OID (pointeur). Cela simplifie les opérations de créations des pointeurs. Sont ainsi apparus NumInstructeurSuiveur, NumDirecteur, NumPayeur, NumAvion au côté de REFInstructeurSuiveur, REFDirecteur, REFPayeur, REFAvion. Dans les types éléments de collection, TypeAvion est au côté de REFT et NumAvion est au côté de REFA.

Dans le Graphe des Types Etendu, on fait apparaître les méthodes qui sont rattachées au type par des liens en pointillés.

Pour respecter l'encapsulation des méthodes, il peut être intéressant de pouvoir accéder aux avions correspondant à un type d'avion à partir de ce type par exemple pour tester qu'un avion fait partie des autonomies ou des qualifications. Sans cette collection de références, il faudrait accéder via les tables objets par une requête SQL. Aussi l'attribut AVIONS de type NT_REF_AVION_TY pour Oracle et de type REF_AVION_TY[] pour PostgreSQL, collection de REF_AVION_TY apparaît au niveau de TYPEAVION_TY.

On obtient le Graphe des Types Complet de l'annexe 4.

Partie 2 : Implantation dans Oracle

Types utilisateurs, collections, références et tables objets

On vous demande d'implanter soit directement via le SQL, soit en éditant le modèle des types de données d'*Oracle SQL Developer Data Modeler* (voir annexe 3), le Graphe des Types Complet de l'annexe 4 en prenant en compte comme tables de stockage objet ADHERENT_OR, INSTRUCTEUR_OR, TYPE_ENGIN_OR, AVION_OR, VOL_OR. Ainsi l'implantation de l'héritage est horizontale : ADHERENT_OR contient toutes les instances de ADHERENT_TY qui ne sont pas dans ses sous-classes. En complément, on crée une table par sous-type, ici INSTRUCTEUR_OR. Les contraintes d'obligation (attribut MANDATORY dans le modèle des types données de d'*Oracle SQL Developer Data Modeler*) de ce document seront implantées dans les tables objet. Les collections seront des *Nested Tables* et auront un index définit sur le référencement par valeur.

A noter que ni dans Oracle, ni dans PostgreSQL, il n'y a la possibilité de créer des variables de classes dans les types utilisateurs. Les attributs de type paramètre seront donc dans une table PARAMETRE_OR de même structure que dans la solution purement relationnelle. Un déclencheur pour Oracle et une fonction utilisateur getSMIC_OR() : n dans un CHECK pour PostgreSQL garantiront des salaires fixes conformes. Des déclencheurs sur la table PARAMETRE_OR gèreront le changement de la valeur du SMIG à la hausse et à la baisse.

Alimentation de la base

Alimentez les tables objets en ne valuant pas dans un premier temps les références. Puis écrire les ordres synchronisant les références avec les valeurs du bi-référencement. Valuez la collection redondante TYPE_ENGIN_OR.Avions. Ecrire un déclencheur AFTER sur la table AVION_OR gérant les évènements INSERT, DELETE, UPDATE sur TypeAERO pour resynchroniser automatiquement Avions.

Implantation des méthodes des types utilisateurs

Dans le Graphe des Types Complet de l'annexe 4, les attributs calculés du dictionnaire des données (annexe 5) seront sous forme de méthodes attachées à un type utilisateur. De même, il existe des méthodes vérifiant la conformité d'un vol et d'un instructeur aux règles de gestions exprimées en annexe 6. L'implantation des méthodes respectera l'encapsulation.

Les types INSTRUCTEUR_TY et VOL_TY ont des méthodes. Ajoutez-les à leur signature et implantez-les dans le corps de chaque type. Les méthodes AAuMoinsUneQualification(), QualificationsDansAutonomies(), BonneDate(), EstAutonome() et EstQualifie() peuvent servir dans des déclencheurs BEFORE INSERT ou UPDATE. Implantez-ces déclencheurs.

Certaines fonctions n'étant opérationnelles que si les références sont synchronisées, les vérifications sont différées. De plus les données déjà chargées ne sont pas toutes conformes pour la table VOL_OR. Créez une table VOL_MAUVAISEDATE_OR et une table VOL_NIAUTONOMENIQUALIFIE_OR qui vont héberger les tuples non conformes.

Gestion de l'unicité de NumAdherent sur deux tables différentes

Dans la solution relationnelle, NumAdherent est gérée par une séquence et l'insertion des nouvelles valeurs passait par l'alimentation de la table ADHERENT d'abord ayant une contrainte d'unicité. Dans la solution objet-relationnelle, il n'y aura pas de séquence pour NumAdherent car NumAdherent et les caractéristiques d'adhérents sont aussi bien présentes dans INSTRUCTEUR_OR que dans ADHERENT_OR. Pour l'alimentation de INSTRUCTEUR_OR, on ignore toute proposition faite, on se contente de proposer une valeur qui ne soit ni dans ADHERENT_OR ni dans INSTRUCTEUR_OR. Pour la table ADHERENT_OR, ce comportement n'est vrai que si la proposition vaut NULL. En cas de proposition déjà présente dans ADHERENT_OR la contrainte de clé primaire fait échouer l'insertion. En cas de proposition déjà présent dans INSTRUCTEUR_OR, il y a correction de la valeur proposée. A noter qu'il y a une séquence pour NumVol et NumInstructeur et donc des déclencheurs qui les gèrent.

Requêtes sur la base de données objet-relationnelle

- Restituer le nom et prénom de toutes les personnes.

- Restituer le nom et prénom de toutes les personnes ainsi que le salaire fixe des instructeurs.
- Restituer les caractéristiques des instructeurs.
- Restituer les caractéristiques des instructeurs avec ses méthodes.
- Restituer les caractéristiques des vols avec ses méthodes.
- Restituer les caractéristiques des vols avec ses méthodes mais aussi les objets liés (directeur, payeur et avion du vol).
- Donnez pour chaque adhérent les caractéristiques de ses autonomies

Partie 3 : Implantation dans PostgreSQL

On vous demande d'implanter la solution proposé dans l'annexe 4 selon la deuxième approche vue en cours en prenant en compte comme tables de stockage objet `ADHERENT_OR`, `INSTRUCTEUR_OR`, `TYPE_ENGIN_OR`, `AVION_OR`, `VOL_OR`. Notamment dans ce graphe, les attributs calculés du dictionnaire des données (annexe 5) seront sous forme de méthodes attachées à un type utilisateur. Les contraintes d'obligation de ce document seront implantées dans les tables objet.

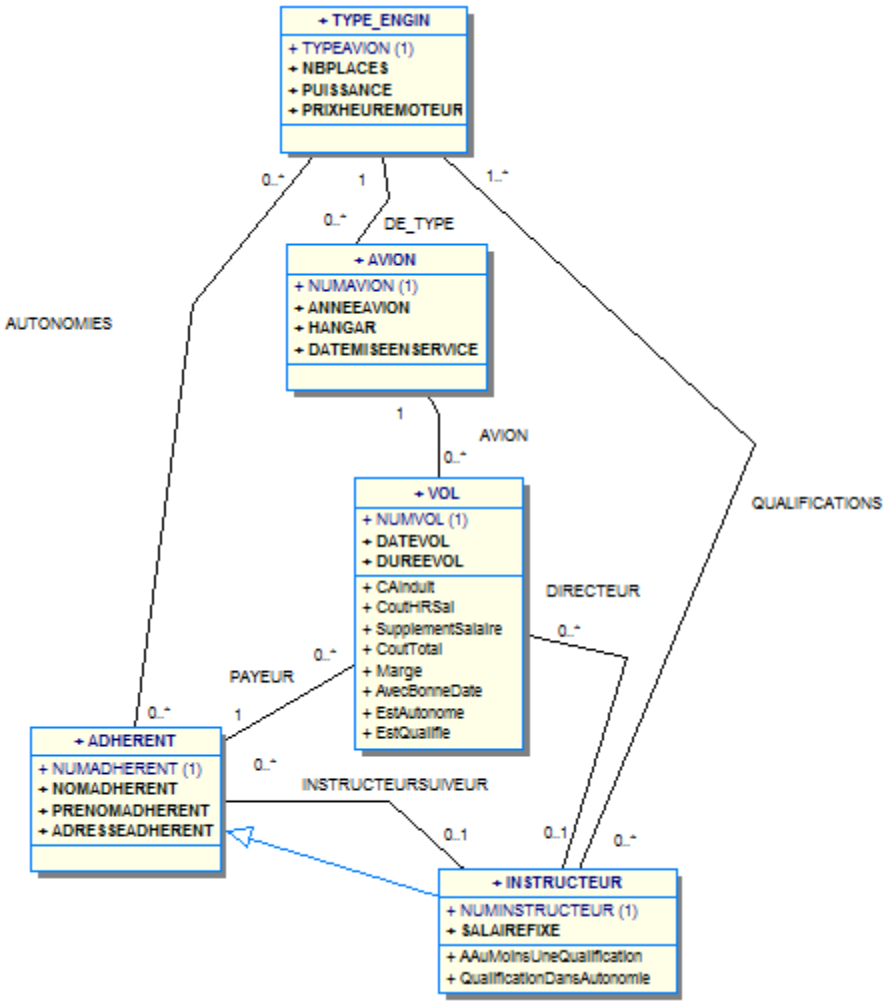
Le Graphe des Types Complet de l'annexe 7 consacré à PostgreSQL est le même que celui d'Oracle (annexe 4) modulo deux différences :

- Le nom des types hors éléments de collection (`REF_TYPE_AVION_OR_TY`, `REF_AVION_OR_TY`) correspond au nom des tables objet-relationnelles de stockage. Ainsi on est en présence de `ADHERENT_OR`, `INSTRUCTEUR_OR`, `TYPE_ENGIN_OR`, `AVION_OR`, `VOL_OR` à la place de `ADHERENT_TY`, `INSTRUCTEUR_TY`, `TYPEAVION_TY`, `AVION_TY`, `VOL_TY`.
- Le type des collections est automatiquement déclaré par PostgreSQL lorsque l'on déclare le type élément de collection en tant que tableau de ce dernier. `REF_AVION_TY[]` remplace `NT_REF_AVION_TY` et `REF_TYPEAVION_TY[]` remplace `NT_REF_TYPEAVION_TY`.

A noter que ni dans Oracle, ni dans PostgreSQL, il n'y a la possibilité de créer des variables de classes dans les types utilisateurs. Les attributs de type paramètre seront donc dans une table PARAMETRE_OR de même structure que dans la solution purement relationnelle. Un déclencheur pour Oracle et une fonction utilisateur `getSMIC_OR()` : `n` dans un CHECK pour PostgreSQL garantiront des salaires fixes conformes. Des déclencheurs sur la table `PARAMETRE_OR` gèreront le changement de la valeur du `SMIG` à la hausse et à la baisse.

On vous demande d'implanter une solution proche de celle implantée dans Oracle en utilisant les fonctions de simulation des références Oracle données en annexe 8. Pour avoir des tables de rejets ayant le même type que les bonnes tables, elles héritent de ces dernières. Aussi les contraintes CHECK doivent tester la table via la colonne `tabloid` pour savoir si les restrictions doivent s'appliquer.

Annexe 1 : Diagramme de classes UML



Annexe 2 : Les relations en 3FN

TYPE_ENGIN(TypeAvion, NBPlaces, Puissance, PrixHeureMoteur)

AVION(NumAvion, #TypeAvion, AnneeAvion, Hangar, DateMiseEnService)

ADHERENT(NumAdherent, #InstructeurSuiveur, NomAdherent, PrenomAdherent, AdresseAdherent)

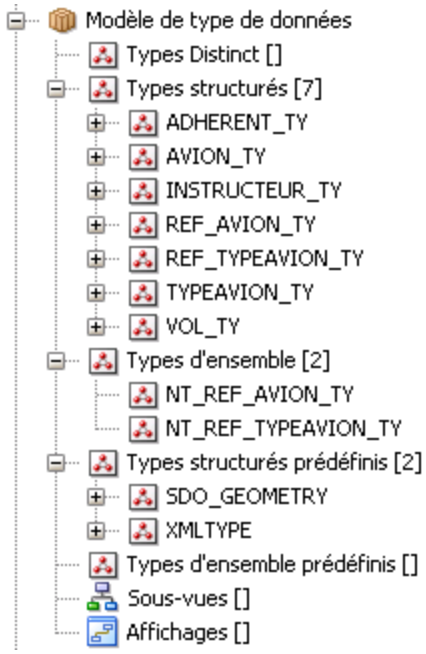
INSTRUCTEUR(NumInstructeur,#NumAdherent, SalaireFixe)

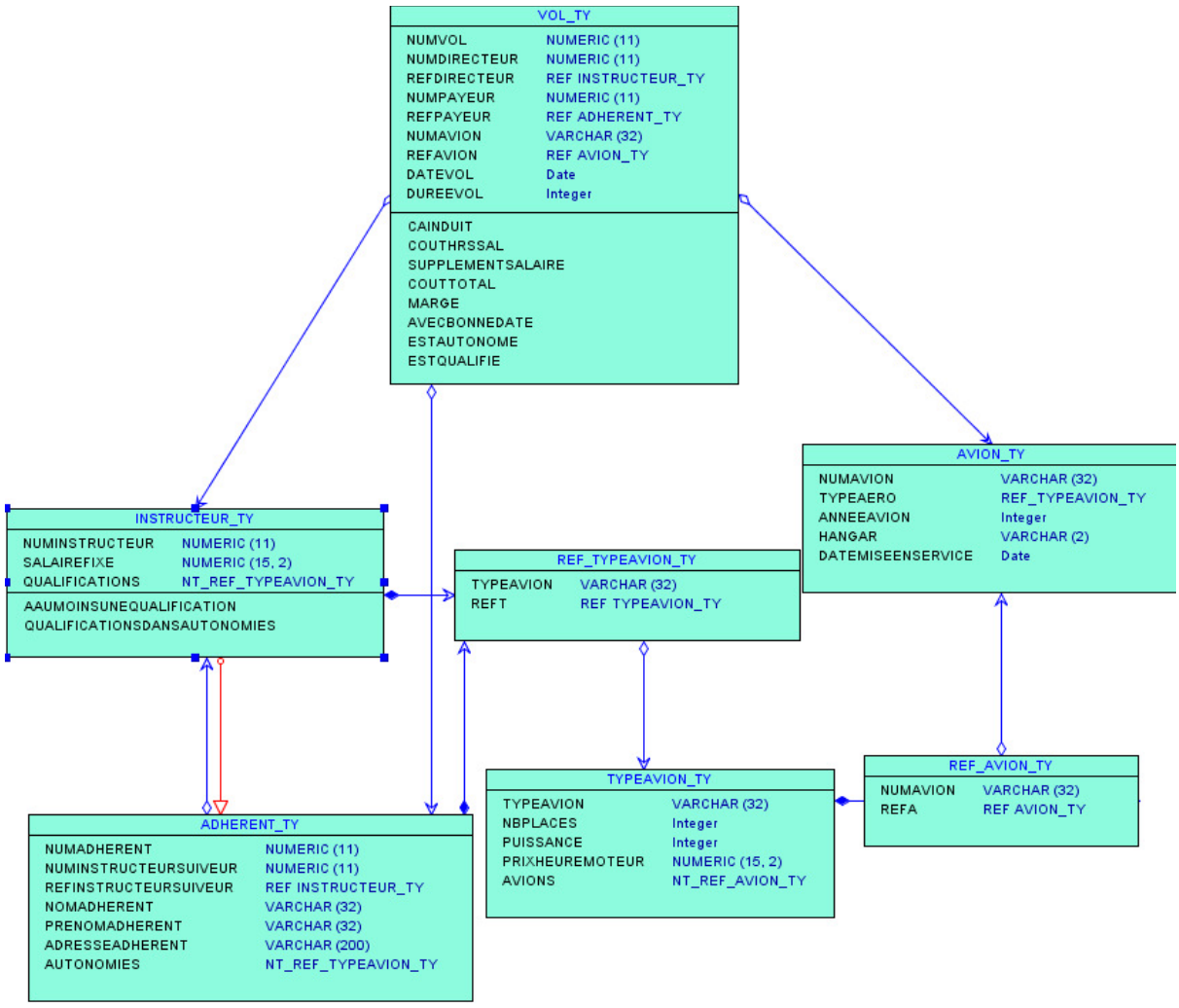
VOL(NumVol, #Directeur, #Payeur, #Avion, DateVol,DureeVol)

AUTONOMIES(#NumAdherent, #TypeAvion)

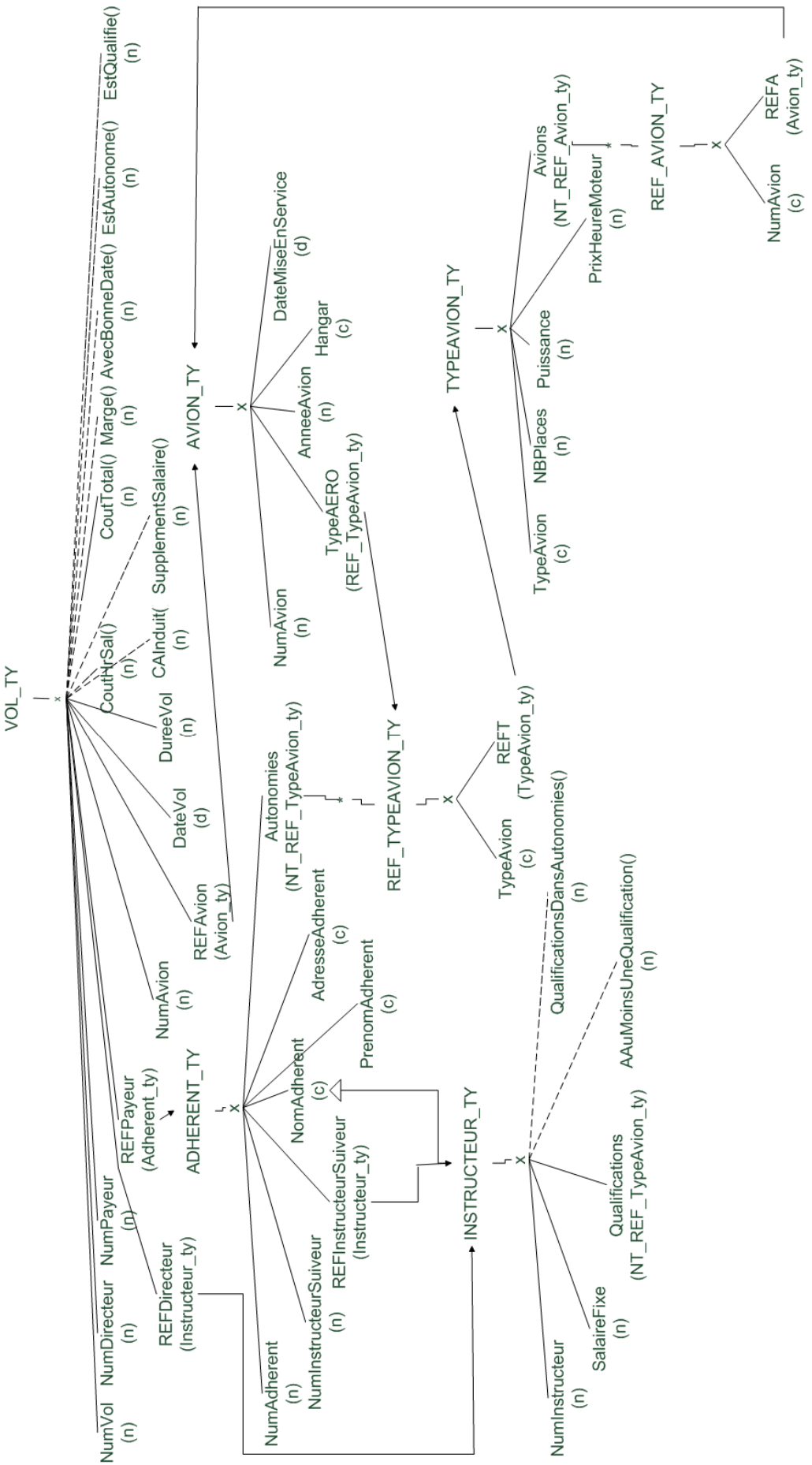
QUALIFICATIONS(#NumInstructeur, #TypeAvion)

Annexe 3 : Les types de données qui peuvent être créé par Oracle SQL Developer Data Modeler





Annexe 4 : Graphe des types complets pour Oracle (voir page suivante)


VOL_TY
NumVol (n)
NumDirecteur (n)
NumPayeur (n)
REFDirecteur (Instructeur_ty)
REFPayeur (Adherent_ty)
NumAvion (n)
REFAvion (Avion_ty)
DateVol (d)
DureeVol (n)
CoutHrSal() (n)
CAInduit((n)
SupplementSalaire() (n)
CoutTotal() (n)
Marge() (n)
AvecBonneDate() (n)
EstAutonome() (n)
EstQualifie() (n)
AVION_TY
NumAvion (n)
DateMiseEnService (d)
Hangar (c)
AnneeAvion (n)
TypeAERO (REF_TypeAvion_ty)
ADHERENT_TY
NumAdherent (n)
NomAdherent (c)
PrenomAdherent (c)
AdresseAdherent (c)
Autonomies (NT_REF_TypeAvion_ty)
REF_TYPEAVION_TY
REFT (TypeAvion_ty)
TypeAvion (c)
INSTRUCTEUR_TY
NumInstructeur (n)
NumInstructeurSuiveur (n)
REFInstructeurSuiveur (Instructeur_ty)
SalaireFixe (n)
Qualifications (NT_REF_TypeAvion_ty)
QualificationsDansAutonomies() (n)
AAuMoinsUneQualification() (n)
TYPEAVION_TY
TypeAvion (c)
NBPlaces (n)
Puissance (n)
Avions (NT_REF_Avion_ty)
PrixHeureMoteur (n)
REF_AVION_TY
NumAvion (c)
REFA (Avion_ty)

Annexe 5 : Dictionnaire des données

Code	Libellé	Nature	Catégorie	Domaine	Commentaire
NUMADHERENT	Numéro de l'adhérent	Incrémentée	Stable	Numérique Obligatoire.	
NOMADHERENT	Nom de l'adhérent	Saisie	Evolutive	Texte(32) Obligatoire.	
PRENOMADHERENT	Prénom de l'adhérent	Saisie	Stable	Texte(32) Obligatoire.	
ADRESSEADHERENT	Adresse de l'adhérent	Saisie	Evolutive	Texte(200) Obligatoire.	
NUMINSTRUCTEUR	Numéro de l'instructeur	Incrémentée	Stable	Numérique Obligatoire.	
SALAIREFIXE	Salaire fixe de l'instructeur	Saisie	Evolutive	Monétaire Obligatoire.	
TYPEAVION	Type d'avion	Saisie	Stable	Texte (32) Obligatoire	
NBPLACES	Nombre de place de l'avion	Saisie	Stable	Numérique Obligatoire	
PUISSANCE	Puissance du moteur	Saisie	Stable	Numérique Obligatoire	
PRIXHEUREMOTEUR	Prix de l'heure du moteur	Saisie	Evolutive	Monétaire Obligatoire	
NUMAVION	Numéro d' immatriculation	Saisie	Stable	Texte(32) Obligatoire	
ANNEEAVION	Année de fabrication	Saisie	Stable	Numérique Obligatoire	
HANGAR	Hangar	Saisie	Stable	Texte (2) Obligatoire. dans « H1, H2 », casse indifférente.	
DATEMISEENSERVICE	Date de mise en service	Saisie	Stable	Date Obligatoire.	

NUMVOL	Numéro du vol	Incrementée	Stable	Numérique Obligatoire	
DATEVOL	Date du vol	Saisie	Stable	DATE Obligatoire Valeur par défaut : date du jour.	
DUREEVOL	Durée du vol	Saisie	Stable	Numérique Obligatoire.	
CAInduit	Chiffre d'affaires induit par le vol	Calculée	Stable	Monétaire Obligatoire.	=DureeVol*PrixHeureMoteur+(350 si directeur)
CoutHrsSal	Coûts hors salaires	Calculée	Stable	Monétaire Obligatoire.	=DureeVol*.8*PrixHeureMoteur
SupplementSalaire	Prime du directeur de vol	Calculée	Stable	Monétaire Obligatoire.	= (300*DureeVol si directeur)
CoutTotal	Coûts induits par le vol	Calculée	Stable	Monétaire Obligatoire.	= CoutHrsSal+ SupplementSalaire
Marge	Marge induite par le vol	Calculée	Stable	Numérique Obligatoire.	=(CAInduit- CoutTotal)/ CAInduit
IdSMIG	Identifiant de la table PARAMETRE_OR	Saisie	Stable	Egal à 1 Obligatoire, UQ	Pas une clé primaire (Pas besoin de jointure naturelle !)
Valsmig	SMIG actuel	Paramètre	Evolutive	Monétaire Non obligatoire	Devra éventuellement être stocké dans une table avec tuple unique pour faciliter la mise à jour de ce paramètre

Annexe 6 : Règles de gestion

En plus du diagramme de classe, on nous indique les contraintes suivantes :

A : Un avion ne peut participer à un vol ayant une date antérieure à sa date de mise en service.

B : Seul un adhérent autonome pour le type d'avion peut voler sans instructeur.

C : Un vol avec un adhérent non autonome a toujours un instructeur. Cet instructeur est unique pour ce vol.

D : Seul un instructeur qualifié pour le type d'avion peut diriger un vol concernant ce même type d'avions.

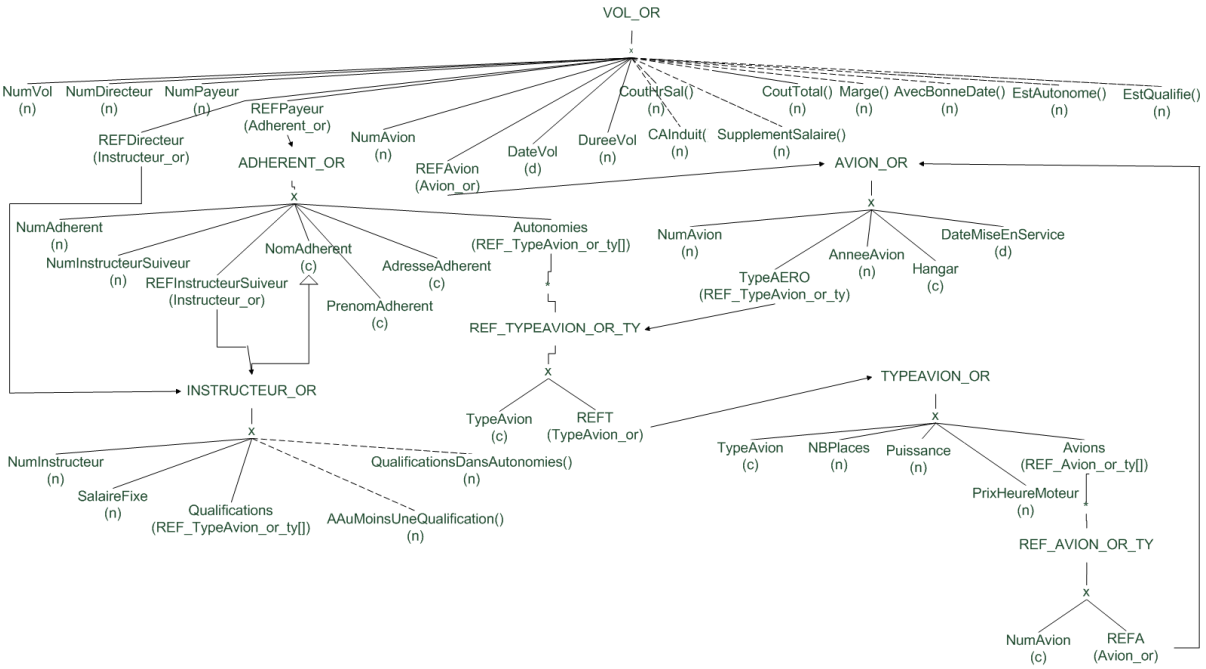
E : Un instructeur qualifié pour un type d'avion est forcément en tant qu'adhérent, autonome pour ce type d'avion

F : Un adhérent qui n'est pas instructeur doit être suivi par un unique instructeur qui sera son interlocuteur privilégié.

G : Aucun salaire n'est inférieur à la valeur du SMIG.

H : Le hangar d'un avion ne peut être que « H1 » ou « H2 », casse indifférente.

Annexe 7 : Graphe des types complets pour PostgreSQL (voir page suivante)



Annexe 8 : Simulation des références pour PostgreSQL

- Le type `TYPE_REF_TY` est déclaré et utilisé par les tables et les types.

```
CREATE TYPE type_ref_ty AS (
  functionref oid,
  objectref oid );
COMMENT ON TYPE type_ref_ty IS 'type pour les références';
```

- Une fonction de génération automatique des fonctions prenant en paramètre l'OID d'un tuple pour le récupérer est déclaré et utilisé lorsque toutes les tables avec OID ont été créées dans le schéma.

```
SELECT generate_getTypeFromOID_function('public');
```

- Des fonctions synchronisant les références à partir des valeurs d'un attribut sont déclarées pour les colonnes valeur et références d'une table, les colonnes valeur et références d'un type utilisé par une colonne d'une table et les colonnes valeur et références d'un type constituant un élément d'une collection d'une colonne d'une table. Toute la collection est synchronisée. Des fonctions simulant la fonction `DEREF` sont automatiquement déclarées au type de la table qui contient la référence.

```
SELECT attribute_sync_ref_from_value('public.adherent_or',
'numinstructeursuiveur', 'refinstructeursuiveur', 'public',
'instructeur_or', 'numinstructeur', 'numinstructeursuiveur is not
null');
SELECT typed_attribute_sync_ref_from_value('public.avion_or',
'typeaero','typeavion', 'REFT','public',
'type_engin_or','typeavion','true');
SELECT array_sync_ref_from_value('public.adherent_or', 'autonomies',
'typeavion', 'REFT','public', 'type_engin_or','typeavion', 'true');
```

Exemple d'utilisation de fonction `DEREF` générée automatiquement :

```
SELECT A.*,getvaluefrom_refinstructeursuiveur(A)
FROM ADHERENT_OR A WHERE numinstructeursuiveur IS NOT NULL;
```

- La fonction `select_object(type_ref_ty, anyelement)` permet de simuler pour les méthodes des type la fonction utilitaire Oracle `utl_ref.select_object`.

Exemple d'utilisation de la fonction dans une methode :

```
CREATE OR REPLACE FUNCTION cainduit(v vol_or)
 RETURNS float AS
$$
DECLARE
  a avion_or%ROWTYPE;
  t type_engin_or%ROWTYPE;
  ca float;
BEGIN
  a:=null;
  t:=null;
  a=select_object(v.refavion,a);
  t=select_object((a).typeaero.reft,t);
```

```
  ca:= (t).prixheuremoteur*(v).dureevol;
  IF v.numdirecteur IS NOT NULL THEN
    ca:=ca+350;
  END IF;
  RETURN ca;
END;
$$
LANGUAGE plpgsql COST 1;
```

Le codes des fonctions :

```
CREATE OR REPLACE FUNCTION generate_getTypeFromOID_function(schemaname text) RETURNS
void AS $func$
DECLARE
    table_keys CURSOR IS
        SELECT c.relname AS f_table_name
        FROM pg_class c, pg_namespace n WHERE c.relhasoids=true and
n.nspname=schemaname AND c.relnamespace = n.oid
        ORDER BY c.relname;
    func_body text;
    func_cmd text;
BEGIN
    FOR table_key IN table_keys LOOP
        RAISE NOTICE 'Genarating for table with oid
%.%',schemaname,table_key.f_table_name;
        func_body := '';
        func_body := func_body ||' SELECT *';
        func_body := func_body ||' FROM '||schemaname||'.'||table_key.f_table_name ;
        func_body := func_body ||' WHERE oid=$1;';
        func_cmd :=
         'CREATE OR REPLACE FUNCTION
getObjectFrom_'||schemaname||'_'||table_key.f_table_name||'(oid) RETURNS
'||schemaname||'.'||table_key.f_table_name||' AS $BODY$'
      || func_body
      || ' $BODY$ LANGUAGE sql COST 1;' ;
        EXECUTE func_cmd;
        func_cmd :='COMMENT ON FUNCTION
getObjectFrom_'||schemaname||'_'||table_key.f_table_name||'(oid)  IS ';
        func_body := 'fonction pour obtenir un objet depuis la table
'||schemaname||'.'||table_key.f_table_name||'.';
        func_cmd :=func_cmd ||quote_literal(func_body)||';';
        EXECUTE func_cmd;
    END LOOP;
END;
$func$ LANGUAGE plpgsql;
COMMENT ON FUNCTION generate_gettypefromoid_function(text) IS 'Procédure générant les
fonction de récupération des objets hors jointure à partir d''une table avec OIDs';

CREATE OR REPLACE FUNCTION
typed_attribute_sync_ref_from_value(fullyQualifiedUpdatedTableName
text,updatedTypedAttributeName text,updatedTypeValueAttributeName
text,updatedTypeREFAttributeName text,schemaNameOfTargetedTable text,
targetedTableName text,keyoftargetedTableName text, optionalUpdateConditions text
default '')
  RETURNS void AS
$BODY1$
DECLARE
  sql_query text;
  result integer:=0;
  func_body text;
  func_cmd text;
```

```
  p_oid oid;
BEGIN
RAISE NOTICE 'Updating table % with typed attribute % (value % and reference %)
referencing table %.%(%) with update condition(s)
%',fullyQualifiedUpdatedTableName,updatedTypedAttributeName,updatedTypeValueAttributeN
ame,updatedTypeREFAttributeName, schemaNameOfTargetedTable, targetedTableName,
keyoftargetedTableName,optionalUpdateConditions ;
IF optionalUpdateConditions<>'' THEN
  optionalUpdateConditions:=' WHERE '||optionalUpdateConditions;
END IF;
sql_query:='UPDATE '||fullyQualifiedUpdatedTableName||' AS T SET ' ||
updatedTypedAttributeName||'.'||updatedTypeREFAttributeName||'.objectref=(SELECT oid
FROM '||schemaNameOfTargetedTable||'.'||targetedTableName||' WHERE
'||keyoftargetedTableName||'=(T.'||updatedTypedAttributeName||').'||updatedTypeValueAt
tributeName||'),'||
updatedTypedAttributeName||'.'||updatedTypeREFAttributeName||'.functionref=(SELECT oid
FROM pg_proc WHERE
proname::name=''getobjectfrom_'||schemaNameOfTargetedTable||'_'||targetedTableName||''
') '||
optionalUpdateConditions;
RAISE NOTICE 'Generated query is: %',sql_query ;
EXECUTE sql_query;
GET DIAGNOSTICS result = ROW_COUNT;
RAISE NOTICE 'Number of updated rows: %',result ;
IF result>0 THEN
  sql_query:='SELECT oid FROM pg_proc WHERE
proname::name=''getobjectfrom_'||schemaNameOfTargetedTable||'_'||targetedTableName||''
'';
  RAISE NOTICE 'Generated query is: %',sql_query ;
  EXECUTE sql_query INTO p_oid;
  RAISE NOTICE 'Generating DEREF function for %.%.% returning
%',fullyQualifiedUpdatedTableName,updatedTypedAttributeName,updatedTypeREFAttributeNam
e,pg_catalog.pg_get_function_result(p_oid);
  func_body := '';
  func_body := func_body ||'DECLARE ';
  func_body := func_body ||'function_name varchar; ';
  func_body := func_body ||'result '||pg_catalog.pg_get_function_result(p_oid)||'; ';
  func_body := func_body ||'BEGIN ';
  func_body := func_body ||'result:=null; ';
  func_body := func_body ||'IF (ty.'||updatedTypedAttributeName||') IS NOT NULL AND
(ty.'||updatedTypedAttributeName||').'||updatedTypeREFAttributeName||' IS NOT NULL AND
(ty.'||updatedTypedAttributeName||').'||updatedTypeREFAttributeName||'.functionref IS
NOT NULL AND
(ty.'||updatedTypedAttributeName||').'||updatedTypeREFAttributeName||'.objectref IS
NOT NULL THEN ';
  func_body := func_body ||'  SELECT proname INTO function_name ';
  func_body := func_body ||'  FROM pg_proc ';
  func_body := func_body ||'  WHERE
oid=(ty.'||updatedTypedAttributeName||').'||updatedTypeREFAttributeName||'.functionref
; ';
  func_body := func_body ||'  EXECUTE ''SELECT * FROM
''||function_name::regproc||''($1)''';
  func_body := func_body ||'  INTO result ';
  func_body := func_body ||'  USING
(ty.'||updatedTypedAttributeName||').'||updatedTypeREFAttributeName||'.objectref; ';
  func_body := func_body ||'END IF; ';
  func_body := func_body ||'return result; ';
  func_body := func_body ||'END; ';
  func_cmd :='CREATE OR REPLACE FUNCTION
getValueFrom_'||updatedTypedAttributeName||'_'||updatedTypeREFAttributeName||' (ty
'||fullyQualifiedUpdatedTableName;
```

```
  func_cmd :=func_cmd ||') RETURNS '||pg_catalog.pg_get_function_result(p_oid)||' AS
$BODY$ '|| func_body|| ' $BODY$ LANGUAGE plpgsql COST 1;' ;
  EXECUTE func_cmd;
  func_cmd :='COMMENT ON FUNCTION
getValueFrom_'||updatedTypedAttributeName||'_'||updatedTypeREFAttributeName||'
('||fullyQualifiedUpdatedTableName||')  IS ';
  func_body := 'fonction DEREF de '||pg_catalog.pg_get_function_result(p_oid)||' pour
la table '||fullyQualifiedUpdatedTableName||'.';
  func_cmd :=func_cmd ||quote_literal(func_body)||';';
  EXECUTE func_cmd;
END IF;
END;
$BODY1$
LANGUAGE plpgsql VOLATILE COST 1;

CREATE OR REPLACE FUNCTION array_sync_ref_from_value(fullyQualifiedUpdatedTableName
text,arrayAttributeName text,updatedNestedTablevalueAttributeName
text,updatedNestedTableREFAttributeName text,schemaNameOfTargetedTable text,
targetedTableName text,keyoftargetedTableName text, optionalUpdateConditions text
default '')
  RETURNS void AS
$BODY1$
DECLARE
  sql_query text;
  max_dim integer:=0;
  result integer:=0;
  global_result integer:=0;
  func_body text;
  func_cmd text;
  p_oid oid;
BEGIN
RAISE NOTICE 'Updating table % array named % [value %, reference %] referencing table
%.%(%) with update condition(s)
%',fullyQualifiedUpdatedTableName,arrayAttributeName,updatedNestedTablevalueAttributeN
ame,updatedNestedTableREFAttributeName, schemaNameOfTargetedTable, targetedTableName,
keyoftargetedTableName,optionalUpdateConditions ;
EXECUTE 'SELECT MAX(array_upper('||arrayAttributeName||',1)) FROM
'||fullyQualifiedUpdatedTableName INTO max_dim;
RAISE NOTICE 'Maximum dim 1 in array named % is: %',arrayAttributeName,max_dim ;
IF optionalUpdateConditions<>'' THEN
  optionalUpdateConditions:=' AND '||optionalUpdateConditions;
END IF;
FOR i IN 1..max_dim LOOP
sql_query:='UPDATE '||fullyQualifiedUpdatedTableName||' AS T SET ' ||
arrayAttributeName||'['||i||'].'||updatedNestedTableREFAttributeName||'.objectref=(SEL
ECT oid FROM '||schemaNameOfTargetedTable||'.'||targetedTableName||' WHERE
'||keyoftargetedTableName||'=T.'||arrayAttributeName||'['||i||'].'||updatedNestedTable
valueAttributeName||'),'||
arrayAttributeName||'['||i||'].'||updatedNestedTableREFAttributeName||'.functionref=(S
ELECT oid FROM pg_proc WHERE
proname::name=''getobjectfrom_'||schemaNameOfTargetedTable||'_'||targetedTableName||''
') '||
'WHERE array_upper(T.'||arrayAttributeName||',1)>='||i||optionalUpdateConditions;
RAISE NOTICE 'Generated query is: %',sql_query ;
EXECUTE sql_query;
GET DIAGNOSTICS result = ROW_COUNT;
RAISE NOTICE 'Number of updated rows: %',result ;
global_result:=global_result+result;
END LOOP;
RAISE NOTICE 'Total number of updated rows: %',global_result ;
IF global_result>0 THEN
```

```
  sql_query:='SELECT oid FROM pg_proc WHERE
proname::name=''getobjectfrom_'||schemaNameOfTargetedTable||'_'||targetedTableName||''
'';
  RAISE NOTICE 'Generated query is: %',sql_query ;
  EXECUTE sql_query INTO p_oid;
  RAISE NOTICE 'Generating DEREF function for array %.%.% returning
%',fullyQualifiedUpdatedTableName,arrayAttributeName,updatedNestedTableREFAttributeNam
e,pg_catalog.pg_get_function_result(p_oid);
  func_body := '';
  func_body := func_body ||'DECLARE ';
  func_body := func_body ||'function_name varchar; ';
  func_body := func_body ||'result '||pg_catalog.pg_get_function_result(p_oid)||'; ';
  func_body := func_body ||'BEGIN ';
  func_body := func_body ||'result:=null; ';
  func_body := func_body ||'IF (ty.'||arrayAttributeName||') IS NOT NULL AND
(ty.'||arrayAttributeName||'[i]).'||updatedNestedTableREFAttributeName||' IS NOT NULL
AND
(ty.'||arrayAttributeName||'[i]).'||updatedNestedTableREFAttributeName||'.functionref
IS NOT NULL AND
(ty.'||arrayAttributeName||'[i]).'||updatedNestedTableREFAttributeName||'.objectref IS
NOT NULL THEN ';
  func_body := func_body ||'  SELECT proname INTO function_name ';
  func_body := func_body ||'  FROM pg_proc ';
  func_body := func_body ||'  WHERE
oid=(ty.'||arrayAttributeName||'[i]).'||updatedNestedTableREFAttributeName||'.function
ref; ';
  func_body := func_body ||'  EXECUTE ''SELECT * FROM
''||function_name::regproc||''($1)''';
  func_body := func_body ||'  INTO result ';
  func_body := func_body ||'  USING
(ty.'||arrayAttributeName||'[i]).'||updatedNestedTableREFAttributeName||'.objectref;
';
  func_body := func_body ||'END IF; ';
  func_body := func_body ||'return result; ';
  func_body := func_body ||'END; ';
  func_cmd :='CREATE OR REPLACE FUNCTION
getValueFrom_'||arrayAttributeName||'_'||updatedNestedTableREFAttributeName||' (ty
'||fullyQualifiedUpdatedTableName;
  func_cmd :=func_cmd ||', i integer) RETURNS
'||pg_catalog.pg_get_function_result(p_oid)||' AS $BODY$ '|| func_body|| ' $BODY$
LANGUAGE plpgsql COST 1;' ;
  EXECUTE func_cmd;
  func_cmd :='COMMENT ON FUNCTION
getValueFrom_'||arrayAttributeName||'_'||updatedNestedTableREFAttributeName||'
('||fullyQualifiedUpdatedTableName||',integer)  IS ';
  func_body := 'fonction DEREF de '||pg_catalog.pg_get_function_result(p_oid)||' pour
la table imbriquée dans la table '||fullyQualifiedUpdatedTableName||'.';
  func_cmd :=func_cmd ||quote_literal(func_body)||';';
  EXECUTE func_cmd;
END IF;
END;
$BODY1$
LANGUAGE plpgsql VOLATILE COST 1;

CREATE OR REPLACE FUNCTION
attribute_sync_ref_from_value(fullyQualifiedUpdatedTableName
text,updatedTablevalueAttributeName text,updatedTableREFAttributeName
text,schemaNameOfTargetedTable text, targetedTableName text,keyoftargetedTableName
text, optionalUpdateConditions text default '')
  RETURNS void AS
$BODY1$
DECLARE
```

```
  sql_query text;
  result integer:=0;
  func_body text;
  func_cmd text;
  p_oid oid;
BEGIN
RAISE NOTICE 'Updating table % with value % and reference % referencing table %.%(%)
with update condition(s)
%',fullyQualifiedUpdatedTableName,updatedTablevalueAttributeName,updatedTableREFAttrib
uteName, schemaNameOfTargetedTable, targetedTableName,
keyoftargetedTableName,optionalUpdateConditions ;
IF optionalUpdateConditions<>'' THEN
  optionalUpdateConditions:=' WHERE '||optionalUpdateConditions;
END IF;
sql_query:='UPDATE '||fullyQualifiedUpdatedTableName||' AS T SET ' ||
updatedTableREFAttributeName||'.objectref=(SELECT oid FROM
'||schemaNameOfTargetedTable||'.'||targetedTableName||' WHERE
'||keyoftargetedTableName||'=T.'||updatedTablevalueAttributeName||'),'||
updatedTableREFAttributeName||'.functionref=(SELECT oid FROM pg_proc WHERE
proname::name=''getobjectfrom_'||schemaNameOfTargetedTable||'_'||targetedTableName||''
') '||
optionalUpdateConditions;
RAISE NOTICE 'Generated query is: %',sql_query ;
EXECUTE sql_query;
GET DIAGNOSTICS result = ROW_COUNT;
RAISE NOTICE 'Number of updated rows: %',result ;
IF result>0 THEN
  sql_query:='SELECT oid FROM pg_proc WHERE
proname::name=''getobjectfrom_'||schemaNameOfTargetedTable||'_'||targetedTableName||''
'';
  RAISE NOTICE 'Generated query is: %',sql_query ;
  EXECUTE sql_query INTO p_oid;
  RAISE NOTICE 'Generating DEREF function for %.% returning
%',fullyQualifiedUpdatedTableName,updatedTableREFAttributeName,pg_catalog.pg_get_funct
ion_result(p_oid);
  func_body := '';
  func_body := func_body ||'DECLARE ';
  func_body := func_body ||'function_name varchar; ';
  func_body := func_body ||'result '||pg_catalog.pg_get_function_result(p_oid)||'; ';
  func_body := func_body ||'BEGIN ';
  func_body := func_body ||'result:=null; ';
  func_body := func_body ||'IF (ty).'||updatedTableREFAttributeName||' IS NOT NULL AND
(ty).'||updatedTableREFAttributeName||'.functionref IS NOT NULL AND
(ty).'||updatedTableREFAttributeName||'.objectref IS NOT NULL THEN ';
  func_body := func_body ||'  SELECT proname INTO function_name ';
  func_body := func_body ||'  FROM pg_proc ';
  func_body := func_body ||'  WHERE
oid=(ty).'||updatedTableREFAttributeName||'.functionref; ';
  func_body := func_body ||'  EXECUTE ''SELECT * FROM
''||function_name::regproc||''($1)''';
  func_body := func_body ||'  INTO result ';
  func_body := func_body ||'  USING (ty).'||updatedTableREFAttributeName||'.objectref;
';
  func_body := func_body ||'END IF; ';
  func_body := func_body ||'return result; ';
  func_body := func_body ||'END; ';
  func_cmd :='CREATE OR REPLACE FUNCTION
getValueFrom_'||updatedTableREFAttributeName||' (ty '||fullyQualifiedUpdatedTableName;
  func_cmd :=func_cmd ||') RETURNS '||pg_catalog.pg_get_function_result(p_oid)||' AS
$BODY$ '|| func_body|| ' $BODY$ LANGUAGE plpgsql COST 1;' ;
  EXECUTE func_cmd;
```

```
  func_cmd :='COMMENT ON FUNCTION getValueFrom_'||updatedTableREFAttributeName||'
('||fullyQualifiedUpdatedTableName||')  IS ';
  func_body := 'fonction DEREF de '||pg_catalog.pg_get_function_result(p_oid)||' pour
la table '||fullyQualifiedUpdatedTableName||'.';
  func_cmd :=func_cmd ||quote_literal(func_body)||';';
  EXECUTE func_cmd;
END IF;
END;
$BODY1$
LANGUAGE plpgsql VOLATILE COST 1;

CREATE OR REPLACE FUNCTION select_object( IN ref type_ref_ty  , INOUT result
anyelement
) AS
$$
DECLARE
function_name varchar;
BEGIN
SELECT proname INTO function_name
FROM pg_proc
WHERE oid=ref.functionref;
EXECUTE 'SELECT * FROM '||function_name::regproc||'($1)'
INTO result
USING ref.objectref;
return;
END;
$$
LANGUAGE plpgsql COST 1;
COMMENT ON FUNCTION select_object(type_ref_ty, anyelement) IS 'Fonction simulant
l''utilitaire utl_ref.select_object disponible dans le PL/SQL d''Oracle';
```