

Modelo relacional

En aquesta pàgina web <http://www.techonthenet.com/oracle/> podem repassar la sintaxi de les sentències bàsiques de SQL i PL/SQL. Oracle enriqueix l'SQL i PL/SQL estàndard per treballar amb objectes. Comencem amb una mica de repàs de l'SQL estàndard respecte a la creació de taules. Els següents exemples, comentats, ens serviran de recordatori.

Exemple: Taula de persones

```
CREATE TABLE Persones
(
    nif varchar2(9) PRIMARY KEY,
    nom varchar2(30),
    cognoms varchar2(80),
    adreça varchar2(120)
);
```

Clau primària simple (formada per un únic atribut). Es pot especificar al costat de l'atribut que és clau primària.

Exemple: Taula de departaments

```
CREATE TABLE Departaments
(
    codi varchar2(5),
    nomdep varchar2(80),
    PRIMARY KEY (codi)
);
```

Clau primària simple (formada per un únic atribut). Es pot especificar després de la definició de tots els atributs. L'avantatge d'aquest mètode és que es poden especificar claus múltiples (formades per més d'un camp) posant els camps separats per comes. El mateix és aplicable per al cas dels altres tipus de restriccions (claus foranes i checs).

Exemple: Taula d'empleats

```
CREATE TABLE Empleats
(
    nif varchar2(9) REFERENCES Persones,
    sou number(6),
    tipus_empleat varchar2(20),
    departament varchar2(5),
    PRIMARY KEY (nif),
    FOREIGN KEY (departament) REFERENCES Departaments ON DELETE CASCADE,
    CHECK (tipus_empleat IN ('Director', 'Treballador'))
);
```

Clau forana simple; es pot especificar la clau forana al costat del camp corresponent com en el cas del nif, o es pot especificar després dels camps com en el cas del departament.

La clàusula ON DELETE CASCADE indica que si s'elimina un departament de la taula de Departaments aleshores s'eliminen tots els empleats d'aquest deparatament de la taula d'Empleats.

El camp tipus_empleat només pot agafar els dos valors especificats en la clàusula CHECK.

Exemple: taula d'estudiants

```
CREATE TABLE Estudiants
(
    nif varchar2(9),
    curs varchar2(10),
    estudis varchar2(50),
    PRIMARY KEY (nif),
    FOREIGN KEY (nif) REFERENCES Persones
);
```

Exemple: taula de matèries

```
CREATE TABLE Materies  
(  
    titolacio varchar2(5),  
    codimat varchar2(5),  
    descripcio varchar2(256),  
    credits number(2),  
    PRIMARY KEY (titolacio,codimat)  
);
```

Aquí tenim un exemple de definició d'una clau primària múltiple. La següent definició **NO** és correcta.

```
CREATE TABLE Materies  
(  
    titolacio varchar2(5) PRIMARY KEY,  
    codimat varchar2(5) PRIMARY KEY,  
    descripcio varchar2(256),  
    credits number(2)  
);
```

Anàlogament passaria amb les claus foranes múltiples.

Sintaxi operacions d'inserció, modificació i eliminació i consultar

INSERT INTO <taula>[(atribut, ... ,atribut)] **VALUES** (valor, ..., valor);

DELETE FROM <taula> [**WHERE** <condició>];

UPDATE <taula> **SET** atribut=valor [,atribut=valor, ... ,atribut=valor] [**WHERE** <condició>];

SELECT {*} | atribut [,atribut, ... ,atribut]] **FROM** <taula [,taula , ... ,taula]> [**WHERE** <condició>]
[**GROUP BY** atribut [,atribut, ... , atribut]] [**HAVING** <condició>] [**ORDER BY** atribut [,atribut, ... , atribut] [asc | desc]];

Model objecte-relacional. Taules d'objectes.

Creació d'una classe. En Oracle és una entitat de tipus Object. La sintaxi per a la creació d'una classe (tipus Object en Oracle) que no és subclasse de cap altre classe és la següent:

```
CREATE [OR REPLACE] TYPE <nom_tipus> AS OBJECT
(
    atribut tipus,
    ...
    ...
    atribut tipus,
    ...
);

```

Les taules poden contenir camps de qualsevol tipus, també de la classe Object. El següent exemple ilustra aquest fet.

Exemple: Creació de les classes Persona_T i la taula Cursos.

```
CREATE TYPE Persona_T as OBJECT
(
    nif varchar2(9),
    nom varchar2(120)
);
CREATE TABLE Cursos
(
    id_curs varchar2(9),
    nom varchar2(120),
    descripcio varchar2(1024),
    professor Persona_T
);
```

En aquest exemple hem creat una taula que té un atribut que és de tipus Objecte (professor és de classe Persona_T).

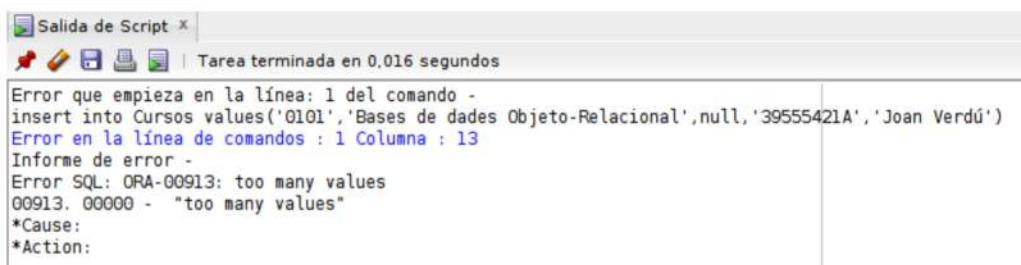
Operacions amb les dedes de les taules: inserció, actualització, selecció i eliminació Inserció

S'utilitza l'operador insert per inserir dades a les taules. La taula Cursos és una taula relacional amb un atribut de la classe Persona_T. Es pot utilitzar o no l'operador new en el valor de l'objecte Persona_T; produeix exactament el mateix resultat. A l'exemple següent les dues instruccions són totalment equivalents.

```
insert into Cursos values('0101','Bases de dades Objeto-Relacional',null,Persona_T('39555421A','Joan Verdú'));
insert into Cursos values('0101','Bases de dades Objeto-Relacional',null, new Persona_T('39555421A','Joan Verdú'));
```

La següent instrucció no s'executarà correctament ja que s'espera un objecte de la classe Persona_T en el quart valor de la clàusula values i es troba un varchar2.

```
insert into Cursos values('0101','Bases de dades Objeto-Relacional',null,'39555421A','Joan Verdú');
```



```
Salida de Script X
Tarea terminada en 0.016 segundos
Error que empieza en la linea: 1 del comando -
insert into Cursos values('0101','Bases de dades Objeto-Relacional',null,'39555421A','Joan Verdú')
Error en la linea de comandos : 1 Columna : 13
Informe de error -
Error SQL: ORA-00913: too many values
00913. 00000 -  "too many values"
*Cause:
*Action:
```

Actualització

Per actualitzar dades s'utilitza l'operador update.

Aquest primer exemple modifica l'atribut nom del camp professor de la taula Cursos.

```
update Cursos C set C.professor.nom = 'Juan Verdú' where C.professor.nif='39555421A';
```

La següent captura de l'sqldeveloper mostra l'estat de la taula després de executar la instrucció d'actualització.

ID_CURS	NOM	DESCRIPCIO	PROFESSOR
0101	Bases de dades Objeto-Relacional	(null)	PROVA.PERSONA_T('39555421A', Juan Verdú')

En aquest altre exemple primer inserirem un curs sense professor i després l'actualitzarem amb un professor.

```
insert into Cursos values('0007','Programació elemental amb JavaFX','GUI's elementals',null);
```

La següent captura de pantalla de l'SQL Developer ens mostra els cursos inserits.

ID_CURS	NOM	DESCRIPCIO	PROFESSOR
0101	Bases de dades Objeto-Relacional	(null)	[PROVA.PERSONA_T]
0007	Programació elemental amb JavaFX	GUI's elementals	(null)

Ara procedim a assignar-le un professor al curs.

```
update Cursos set professor = Persona_T('18123759X','Manuel Rivas') where id_curs='0007';
```

Comprovem que s'han fet els canvis.

ID_CURS	NOM	DESCRIPCIO	PROFESSOR
0101	Bases de dades Objeto-Relacional	(null)	[PROVA.PERSONA_T]
0007	Programació elemental amb JavaFX	GUI's elementals	PROVA.PERSONA_T('18123759X','Manuel Rivas')

Selecció

Per fer consultes s'utilitzen àlies i la notació separada pels punts per accedir als atributs dels objectes. Per exemple la consulta que mostra el nom dels cursos i el nif i nom dels professors que els imparteixen es faria de la següent manera:

```
select C.nom as NomCurs, C.professor.nif as NIF, C.professor.nom as NomProfessor from Cursos C;
```

NOMCURS	NIF	NOMPROFESSOR
Bases de dades Objeto-Relacional	39555421A	Juan Verdú
Programació elemental amb JavaFX	18123759X	Manuel Rivas

Si fem un select de tots els cursos des del SQL Developer ens dóna el següent resultat.

```
select * from Cursos;
```

ID_CURS	NOM	DESCRIPCIO	PROFESSOR
0101	Bases de dades Objeto-Relacional	(null)	PROVA.PERSONA_T('39555421A', 'Juan Verdú')
0007	Programació elemental amb JavaFX	GUI's elementals	PROVA.PERSONA_T('18123759X', 'Manuel Rivas')

Vaig a inserir el curs de 'Programació en PL/SQL' impartit per Juan Verdú.

```
insert into Cursos values('0102','Programació en PL/SQL',null,Persona_T('39555421A','Juan Verdú'));
```

Treballant amb objectes també es poden fer agrupacions per atribut d'un objecte i es poden emprar les clàusules `order by`. A continuació es mostra un exemple d'utilització de l'agrupació i ordenació.

La següent consulta mostra quants cursos imparteix cada persona ordenats descendentalment per la quantitat de cursos impartits. Fixem-nos en què estem agrupant per un camp que és un atribut de l'objecte professor (de la classe `Persona_T`).

```
select C.professor.nif as NIF, C.professor.nom as Nom, count(*) as NumCursos from Cursos C  
group by C.professor.nif, C.professor.nom order by NumCursos;
```

NIF	NOM	NUMCURSOS
18123759X	Manuel Rivas	1
39555421A	Juan Verdú	2

Eliminació

L'eliminació de dades es fa amb l'operador `delete`.

Es poden eliminar registres utilitzant els valors dels atributs de camps tipus Objecte en la condició de la instrucció. En el següent exemple eliminarem els cursos impartits per professors el nif dels quals comenci per 18. Segons les dades que tenim, això eliminaria el curs impartit per Manuel Rivas.

```
delete from Cursos C where C.professor.nif like '18%';
```

Comprovem que la instrucció ha fet el que esperàvem mitjançant una consulta.

```
select nom,professor from Cursos C;
```

NOM	PROFESSOR
Bases de dades Objeto-Relacional	PROVA.PERSONA_T('39555421A', 'Juan Verdú')
Programació en PL/SQL	PROVA.PERSONA_T('39555421A', 'Juan Verdú')

Taules d'objectes

La taula de Cursos amb la qual hem estat treballant és una taula relacional que conté un camp (atribut de la taula) que és un objecte; el camp professor.

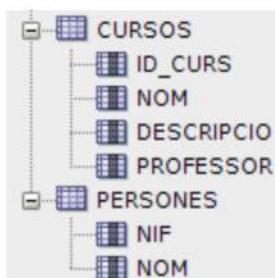
Ara definirem un altre tipus de taules especials que es defineixen com a taules d'objectes. La definició d'una taula d'objectes segueix la següent sinatxi:

```
CREATE TABLE <taula> OF <tipus_objecte>;
```

Les taules que es creen d'aquesta manera es poden veure des de una perspectiva relacional o des de la perspectiva de l'orientació a objectes.

Anem a crear una taula Persones d'objectes Persona_T. Seguim la sintaxi indicada en la línia de dalt hauríem de fer:

```
CREATE TABLE Persones OF Persona_T;
```



La següent captura de la vista de l'estruccura de les taules en SQL Developer mostra com en el cas de la taula Cursos apareix l'atribut professor que és de la classe Persona_T i com a la taula Persones no apareix cap atribut Persona_T. Sembla com si aquesta taula fos una taula relacional amb 2 camps: nif i nom. Realment és una taula d'objectes en la qual podrem inserir dades com si es tractés d'una taula purament relacional o bé inserir objectes de la classe Persona_T. El següents exemples aclariran una mica aquests conceptes.

Inserirem 2 persones a la taula Persones.

En la primera inserció inserirem les dades des de la perspectiva relacional:

```
insert into Persones values('39222277W','Casimiro Miró');
```

En la segona inserció inserirem les dades des de la perspectiva orientada a objectes:

```
insert into Persones values(Persona_T('25111259R','Ana Mohedano'));
```

Consultar totes les persones de la taula tal com ho fem habitualment fa la consulta de la taula des de la perspectiva purament relacional.

```
select * from Persones;
```

NIF	NOM
39222277W	Casimiro Miró
25111259R	Ana Mohedano

Per mostrar el resultat de la consulta des de la perspectiva de l'orientació a objectes necessitem l'operador **VALUE**. La mateixa consulta des d'aquesta perspectiva es faria així:

```
select value(P) as Persona from Persones P;
```

Captura de l'SQL Developer

PERSONA
PROVA.PERSONA_T('39222277W','Casimiro Miró')
PROVA.PERSONA_T('25111259R','Ana Mohedano')

Captura de l'SQL Plus

```
PERSONA(NIF, NOM)
-----
PERSONA_T('39222277W', 'Casimiro Miró')
PERSONA_T('25111259R', 'Ana Mohedano')
```

Ara vaig a afegir a la Pilar Esteve amb NIF 39222277W.

```
insert into Persones values(Persona_T('39222277W','Pilar Esteve'));
```

Que ens mostra la següent captura de pantalla ?

NIF	NOM
39222277W	Casimiro Miró
25111259R	Ana Mohedano
39222277W	Pilar Esteve

La resposta és evident. Tenint 2 persones amb el mateix NIF; la Pilar i en Casimiro. L'Oracle s'ha empassat això. Afortunadament això es pot evitar definitivament en la taula de persones definint el nif com a clau primària. Es defineix així (prèviament esborro a Pilar Esteve per no tenir problemes a l'hora de definir la clau primària):

```
alter table Persones add primary key (nif);
```

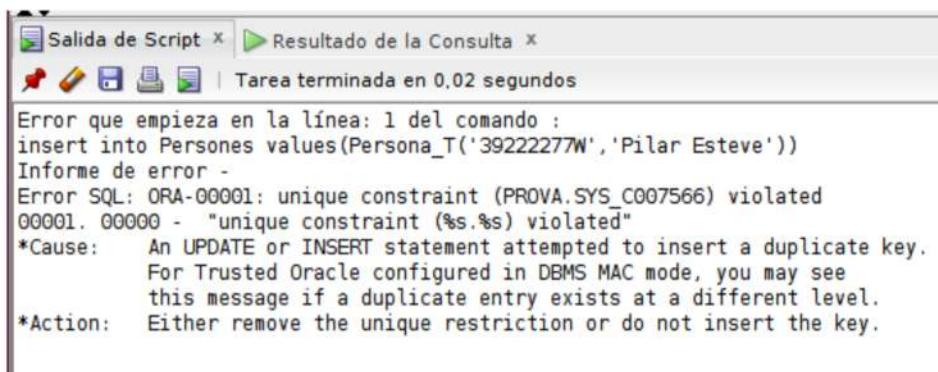
Quan vam crear iniciament la taula Persones havíem d'haver tingut en compte la integritat de clau primària. La taula s'havia d'haver creat d'aquesta manera:

```
CREATE TABLE Persones OF Persona_T
(
    primary key (nif)
);
```

Ara tractaré d'inserir altre cop a Pilar Esteve amb el mateix nif que Casimiro Miró.

```
insert into Persones values(Persona_T('39222277W','Pilar Esteve'));
```

L'SQL Developer em dóna aquest error:



```
Salida de Script x Resultado de la Consulta x
| Tarea terminada en 0.02 segundos
Error que empieza en la linea: 1 del comando :
insert into Persones values(Persona_T('39222277W','Pilar Esteve'))
Informe de error -
Error SQL: ORA-00001: unique constraint (PROVA.SYS_C007566) violated
00001. 00000 - "unique constraint (%s.%s) violated"
*Cause: An UPDATE or INSERT statement attempted to insert a duplicate key.
For Trusted Oracle configured in DBMS MAC mode, you may see
this message if a duplicate entry exists at a different level.
*Action: Either remove the unique restriction or do not insert the key.
```

on clarament, encara que sigui en anglès, ens indica que aquest nif ja existeix.

Aquestes taules, a tots els efectes, són com les taules relacionals pures i es poden fer totes les operacions que es fan amb aquestes últimes. Aquestes taules tenen un identificador (OID) per a cada fila (objecte) inserit que si vulguessim podríem utilitzar com a clau primària.

Taules d'objectes amb herència i polimorfisme

Una forma d'implementar l'herència en el model relacional consisteix en definir una taula per a la superclasse amb els atributs comuns i una taula per cada subclasse amb els atributs específics de cada subclasse de manera que les claus primàries de totes les taules són les mateixes i cada clau primària en cada subclasse és a més clau forana referenciant a la taula de la superclasse.

La sintaxi per implementar la integritat referencial en una taula d'objectes (des de la perspectiva del model relacional) és molt semblant a la utilitzada en una taula sense objectes. Aquesta seria la sitaxi:

```
CREATE TABLE <taula> OF <tipus_objecte>
(
    atribut-clau tipus,
    [atribut-clau tipus,
    ...
    atribut-clau tipus,]
    ...
    atribut tipus,
    ...
    PRIMARY KEY (atribut-clau[,atribut-clau,...,atribut-clau]),
    FOREIGN KEY (atribut[,atribut,...,atribut]) REFERENCES <taula>
);

```

Exemple

Definirem les classes Estudiant_T i Professor_T les quals les considerarem subclasses de Persona_T. A continuació es definirà una taula d'objectes per cada classe diferent; s'especificaran les claus primàries i foranes segons les regles comentades en les línies de dalt.

```
CREATE TYPE Estudiant_T as OBJECT
(
    nif varchar2(9),
    estudis varchar2(40),
    curs number(2)
);
CREATE TABLE Estudiants of Estudiant_T
(
    primary key (nif),
    foreign key (nif) references Persones
);
```

```
CREATE TYPE Professor_T as OBJECT
(
    nif varchar2(9),
    titol varchar2(60)
);
CREATE TABLE Professors of Professor_T
(
    primary key (nif),
    foreign key (nif) references Persones
);
```

Amb això hem aconseguit la integritat referencial. Podíem haver afegit una clàusula `on delete cascade` en les subclasses de manera que quan s'esborri una persona s'esborri automàticament aquesta persona de la taula d'estudiants o professors.

Quan executo la següent instrucció es produeix un error degut a la integritat referencial; no existeix la persona amb nif 39672153B.

```
insert into Estudiants values(Estudiant_T('39672153B','ASIX',2));
```

```
Error que empieza en la linea: 1 del comando :
insert into Estudiants values(Estudiant_T('39672153B', 'ASIX',2))
Informe de error -
Error SQL: ORA-02291: integrity constraint (PROVA.SYS_C007569) violated - parent key not found
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"
*Cause:    A foreign key value has no matching primary key value.
*Action:   Delete the foreign key or add a matching primary key.
```

Si inserim primer com a persona a l'estudiant de nif 39672153B i després tornem a fer l'insert d'abans aleshores si que funciona.

```
insert into Persones values(Persona_T('39672153B','Roger Benaiges'));
insert into Estudiants values(Estudiant_T('39672153B','ASIX',2));
```

Després de les instruccions tenim un estudiant a la taula d'estudiants.

NIF	ESTUDIS	CURS
39672153B	ASIX	2

Amb aquest enfocament no aprofitem la potència dels mecanismes d'herència propis dels llenguatges orientats a objectes.

A partir de la versió 9, Oracle incorpora els mecanismes d'herència que utilitza qualsevol llenguatge orientat a objectes on, per exemple, es podran definir mètodes i sobreescrivir els mètodes de la superclasse.

Per fer això es necessita definir la superclasse com **NOT FINAL** (per defecte es **FINAL**) que ens permetrà derivar subclasses a partir d'aquesta. Després s'ha d'indicar a cada classe quina és la seva superclasse amb la clàusula **UNDER**. La sintaxi general per fer això seria així:

```
CREATE [OR REPLACE] TYPE <super-tipus> AS OBJECT
```

```
(  
    atribut tipus,  
    ...  
    atribut tipus,  
    ...  
) [FINAL|NOT FINAL];
```

```
CREATE [OR REPLACE] TYPE <sub-tipus> UNDER <super-tipus>
```

```
(  
    atribut tipus,  
    ...  
    atribut tipus,  
    ...  
) [FINAL|NOT FINAL];
```

```
CREATE TABLE <taula> OF <super-tipus>
```

```
(  
    atribut-clau tipus,  
    [atribut-clau tipus,  
    ...  
    atribut-clau tipus,]  
    ...  
    atribut tipus,  
    ...  
    PRIMARY KEY (atribut-clau[,atribut-clau,...,atribut-clau]))  
);
```

Amb els mecanismes d'herència orientat a objectes i la integritat referencial del model relacional tornem a definir les taules de estudiants i professor.

Primer modificarem la classe Persona_T per fer-la **NOT FINAL**.

```
alter type Persona_T not final cascade;
```

A continuació modificarem les subclasses per fer que siguin subclasses. El que farem serà eliminar-les i tornar-les a crear.

Si ja sabíem des del començament que s'heredaria de la classe Persona_T hauria estat millor opció crear la classe indicant que és una classe de la qual es pot heredar.

```
CREATE TYPE Persona_T as OBJECT
```

```
(  
    nif varchar2(9),  
    nom varchar2(120)  
) NOT FINAL;
```

Ara hem de crear les subclasses de Persona_T (esborro les classes Estudiant_T i Professor_T que havia creat abans per tornar-los a definir).

```
CREATE TYPE Estudiant_T UNDER Persona_T  
(  
    estudis varchar2(40),  
    curs number(2)  
)
```

```
CREATE TABLE Estudiants of Estudiant_T  
(  
    primary key (nif),  
    foreign key (nif) references Persones  
)
```

```
CREATE TYPE Professor_T UNDER Persona_T  
(  
    titol varchar2(60)  
)
```

```
CREATE TABLE Professors of Professor_T  
(  
    primary key (nif),  
    foreign key (nif) references Persones  
)
```

Ja tinc definida l'herència des de la perspectiva de l'orientació a objectes amb la integritat referencial. Netejo la taula de persones (sense cap dada).

Provem d'inserir a l'estudiant Roger.

```
insert into Estudiants values(Estudiant_T('ASIX',2));
```

```
Error que empieza en la linea: 1 del comando -  
insert into Estudiants values(Estudiant_T('ASIX',2))  
Error en la linea de comandos : 1 Columna : 31  
Informe de error -  
Error SQL: ORA-02315: numero incorrecto de argumentos para el constructor por defecto  
02315. 00000 - "incorrect number of arguments for default constructor"  
*Cause: The number of arguments specified for the default constructor  
doesn't match the number of attributes of the object type.  
*Action: Specify the correct number of arguments for the default  
constructor and retry the operation.
```

Necessitem cridar al constructor de l'estudiant amb els quatre paràmetres; els definits a la classe més els definits a la seva superclasse.

```
insert into Estudiants values(Estudiant_T('39672153B','Roger Benaiges','ASIX',2));
```

```
Error que empieza en la linea: 1 del comando :  
insert into Estudiants values(Estudiant_T('39672153B','Roger Benaiges','ASIX',2))  
Informe de error -  
Error SQL: ORA-02291: integrity constraint (PROVA.SYS_C007575) violated - parent key not found  
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"  
*Cause: A foreign key value has no matching primary key value.  
*Action: Delete the foreign key or add a matching primary key.
```

Es produeix un error d'integritat referencial. Primer hem d'introduir-lo a la taula Persones.

```
insert into Persones values(Persona_T('39672153B','Roger Benaiges'));
```

Ara ja el puc introduïr a Estudiants.

```
insert into Estudiants values(Estudiant_T('39672153B','Roger Benaiges','ASIX',2));
```

L'estudiant s'ha inserit correctament a la taula Estudiants.

NIF	ESTUDIS	CURS	NOM
39672153B	ASIX	2	Roger Benaiges

Si en el model de dades d'una aplicació tenim que les subclasses d'una classe només es relacionen amb la seva superclasse mitjançant herència i no es relacionen amb cap altra classe aleshores la manera més eficient de treballar és utilitzar una única taula d'objectes del tipus de la superclasse on s'emmagatzemaran tant objectes de la superclasse com de les seves subclasses.

Per exemple si els estudiants i professors només es relacionen mitjançant herència amb les persones només caldria tenir la taula de persones a la qual podríen afegir persones, professors i estudiants.

Crearem una taula anomenada Persones2 de Persona_T i hi afegirem dades.

```
CREATE TABLE Persones2 of Persona_T
```

```
(  
primary key (nif)  
);
```

```
insert into Persones2 values(Persona_T('39222277W','Pilar Esteve'));
```

```
insert into Persones2 values(Estudiant_T('39672153B','Roger Benaiges','ASIX',2));
```

```
insert into Persones2 values(Professor_T('39541275M','Gisela Carretero','Informàtica'));
```

Les 3 instruccions d'inserció han inserit les dades correctament a la taula Persones2.

La següent captura de pantalla de l'SQL Developer mostra les dades de la taula Persones2.

NIF	NOM
1 39222277W	Pilar Esteve
2 39672153B	Roger Benaiges
3 39541275M	Gisela Carretero

Podem comprovar que només es veuen els valors dels camps de la classe Persona_T i no els valors de les subclasses Estudiant_T i Professor_T. Mirarem de fer algunes consultes que ens mostrin aquests camps.

Consulta elemental de totes les dades de la taula Persones2.

```
select * from Persones2;
```

NIF	NOM
39222277W	Pilar Esteve
39672153B	Roger Benaiges
39541275M	Gisela Carretero

Tornen a sortir només els valors dels camps de la classe Persona_T (perspectiva relacional).

```
select value(P) from Persones2 P;
```

```
-----  
VALUE(P)  
-----  
PROVA.PERSONA_T('39222277W','Pilar Esteve')  
PROVA.ESTUDIANT_T('39672153B','Roger Benaiges','ASIX',2)  
PROVA.PROFESSOR_T('39541275M','Gisela Carretero','Informàtica')
```

Ara sí que em surten bé les dades de totes les persones independentment que sigui estudiant, professor o cap de les dues.

Si vull només recuperar totes les dades dels estudiants (una de les subclasses) puc utilitzar l'operador **ONLY** conjuntament amb l'operador **VALUE**.

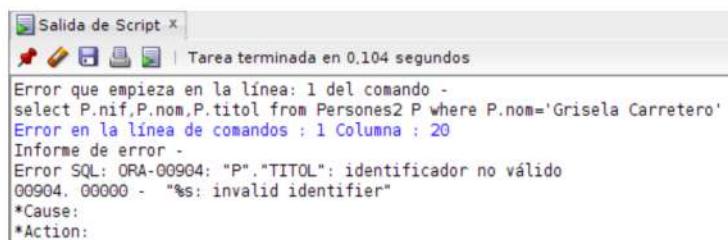
select value(P) from Persones2 P where value(P) is of (ONLY Estudiant_T);

```
VALUE(P)
-----
PROVA.ESTUDIANT_T('39672153B', 'Roger Benaiges', 'ASIX', 2)
```

La següent consulta recupera les dades de les persones que són estudiants.

Consulta de les dades de la professora Grisela Carretero.

select P.nif,P.nom,P.titol from Persones2 P where P.nom='Grisela Carretero';



M'està donant un error dient-me que el camp **títol** no existeix. El problema és que no és pot accedir directament als atributs (i mètodes) de la superclasse. Necessitem utilitzar l'operador **TREAT**.

Solució utilitzant l'operador **TREAT**.

select VALUE(P).nif as NIF,VALUE(P).nom as NOM,TREAT(VALUE(P) as Professor_T).titol as TITOL from Persones2 P where VALUE(P).nom='Gisela Carretero';

NIF	NOM	TITOL
39541275M	Gisela Carretero	Informàtica

Aquesta mateixa instrucció es pot posar de forma més compacta:

select nif,nom,TREAT(VALUE(P) as Professor_T).titol as TITOL from Persones2 P where nom='Gisela Carretero';

L'operador **TREAT** treballa conjuntament amb l'operador **VALUE** i em serveix per obtenir els atributs (i/o mètodes) de les subclasses forçant a interpretar el valor obtingut amb **VALUE** com de la classe **Estudiant_T**.

L'operador **TREAT** retorna **NULL** quan opera sobre objectes que no són del tipus especificat. Això pot provocar que el resultat de la consulta tingui "forats" tal com es mostra al següent exemple. Aquest problema es pot solucionar fàcilment utilitzant també l'operador **ONLY**.

A continuació insereixo un altre estudiant.

insert into Persones2 values(Estudiant_T('25111259R','Ana Mohedano','DAW',1));

La següent instrucció mostraria tots els estudiants pero deixaria forats.

select TREAT(VALUE(P) as Estudiant_T) from Persones2 P;

```
TREAT(VALUE(P) as ESTUDIANT_T)
-----
PROVA.ESTUDIANT_T('39672153B', 'Roger Benaiges', 'ASIX', 2)
PROVA.ESTUDIANT_T('25111259R', 'Ana Mohedano', 'DAW', 1)
```

Executem la mateixa però afegint-li l'operador ONLY.

```
select TREAT(VALUE(P) as Estudiant_T) from Persones2 P where value(P) is of (ONLY Estudiant_T);
```

```
TREAT(VALUE(P) as ESTUDIANT_T)
-----
PROVA.ESTUDIANT_T('39672153B', 'Roger Benaiges', 'ASIX', 2)
PROVA.ESTUDIANT_T('25111259R', 'Ana Mohedano', 'DAW', 1)
```

Es pot comprovar com s'han eliminat les línies en blanc.

Referències

A una taula es poden definir camps que són de tipus referència. Cada objecte en una taula d'objectes té un identificador anomenat OID que l'identifica de manera unívoca.

Nosaltres podem definir atributs en una classe que siguin precisament una referència a un objecte (OID) d'una altra taula. Així mitjançant els tipus referència podem implementar els diferents tipus de relacions entre taules.

Exemple: Implementació de la relació entre empleats i els departaments als quals pertanyen.

Suposarem que la navegabilitat és únicament des de l'empleat cap al departament; és a dir no ens interessa tenir implementades referències des de departaments cap a empleats.

Pas 1: Definir les classes per als empleats i els departaments.

```
CREATE TYPE Departament_T as OBJECT
(
    codi varchar2(4),
    nom varchar2(80)
);
```

```
CREATE TYPE Empleat_T as OBJECT
(
    nif varchar2(9),
    nom varchar2(80),
    departament REF Departament_T
);
```

Pas 2: Creació de les taules d'empleats i departaments.

```
CREATE TABLE Departaments of Departament_T
(
    primary key (codi)
);
```

```
CREATE TABLE Empleats of Empleat_T
(
    primary key (nif)
);
```

Pas 3: Inserció de dades en les taules.

```
insert into Departaments values(Departament_T('C001','Comptabilitat'));
```

Com fem ara per a inserir un empleat que pertanyi al departament de comptabilitat ?

Solució: Utilitzant l'operador REF. L'operador REF retorna l'OID dels objectes d'una taula d'objectes (un OID per filera). Faig una consulta de l'OID del departament de codi C001; el de comptabilitat.

```
select REF(P) from Departaments P where codi='C001';
```

Des de l'SQL Developer no es mostra aquest OID quan faig la consulta.

```
REF(P)
-----
PROVA.DEPARTAMENT_T('C001','Comptabilitat')
```

En canvi si faig la mateixa consulta des de l'SQL Plus si ens apareix la referència.

```
SQL> select REF(P) from Departaments P where P.codi='C001';
REF(P)
-----
0000280209F38FB5883234A55FE040007F01010600F38FB5883232A55FE040007F01010600010004A60000
```

Per tant per inserir l'empleat només hem de posar tot aquest select com a 3r valor a l'insert.

```
insert into Empleats values(Empleat_T('39672153B','Viktòria Montalvo',(select REF(P) from Departaments P where codi='C001')));
```

Inserirem altre empleat al mateix departament i mostrarem tots els departaments que tenen empleats.

```
insert into Empleats values(Empleat_T('21188594L','David Peñalver',(select REF(P) from Departaments P where P.codi='C001')));
```

```
select departament from Empleats;
```

```
DEPARTAMENT
```

```
-----  
PROVA.DEPARTAMENT_T('C001','Comptabilitat')  
PROVA.DEPARTAMENT_T('C001','Comptabilitat')
```

Podem veure que des de l'SQL Developer es desreferèncien automàticament les referències. A més ens han sortit el departament duplicat, cosa que s'elimina amb facilitat ja que hem de fer un group by per obtenir aquests departaments.

```
select departament from Empleats group by departament;
```

```
DEPARTAMENT
```

```
-----  
PROVA.DEPARTAMENT_T('C001','Comptabilitat')
```

Amb l'SQL Plus sí que ens surt la referència.

```
SQL> select departament from empleats group by departament;  
DEPARTAMENT  
-----  
0000220208F38FB5883234A55FE040007F01010600F38FB5883232A55FE040007F01010600
```

Si tractem d'inserir un empleat a un departament i el select de la referència que fem sobre la taula Departaments és buit aleshores s'insereix **null** al departament de l'empleat.

Com fem per mostrar els departaments i no les referències ?

Solució: Utilitzar l'operador **DEREF** per desreferenciar les referències. La següent consulta mostra els departaments que tenen empleats (sense repeticions).

```
select DEREF(departament) as Departaments from empleats group by departament;
```

```
SQL> select DEREF(departament) as Departaments from empleats group by departament;  
DEPARTATMETS(CODI, NOM)  
-----  
DEPARTAMENT_T('C001', 'Comptabilitat')
```

Integritat referencial

A continuació vaig a esborrar el departament de comptabilitat.

```
delete from Departaments where codi = 'C001';
```

Resulta que la consulta esborra el departament de comptabilitat i deixa als empleats amb referències penjades (apunten a un objecte que ja no existeix). Una referència penjada es diu que està en estat **DANGLING**.

La manera de solucionar-ho és definir una clau forana en el camp que té la referència.

Podem optar per modificar l'estructura de la taula Empleats i afegir-li la clau forana (cal que la taula d'empleats estigui buida per fer això) o bé tornar a crear les taules de nou.

Modificar taula	ALTER TABLE Empleats add foreign key (departament) references Departaments;
Crear taula	CREATE TABLE Empleats of Empleat_T (primary key (nif), foreign key (departament) references Departaments);

Tornem a inserir les dades

```
insert into Departaments values(Departament_T('C001','Comptabilitat'));
insert into Empleats values(Empleat_T('39672153B','Víctoria Montalvo',(select REF(P) from Departaments P where codi='C001')));
insert into Empleats values(Empleat_T('21188594L','David Peñalver',(select REF(P) from Departaments P where P.codi='C001')));
```

Ara quan tractem d'esborrar el departament ens lanza un error degut a la violació de la integritat referencial.

```
delete from Departaments where codi = 'C001';
```

```
Error que empieza en la linea: 1 del comando :
delete from Departaments where codi = 'C001'
Informe de error -
Error SQL: ORA-02292: integrity constraint (PROVA.SYS_C007587) violated - child record found
02292. 00000 - "integrity constraint (%s.%s) violated - child record found"
*Cause:    attempted to delete a parent key value that had a foreign
          dependency.
*Action:   delete dependencies first then parent or disable constraint.
```

A més la restricció d'integritat referencial força a que les referències de la taula d'empleats siguin referències vàlides en la taula de Departaments per la qual cosa no cal definir la clàusula SCOPE en la columna amb la referència. La clàusula SCOPE serveix per forçar a que les referències siguin a elements d'una taula concreta. Per comprovar la restricció d'àmbit he creat una taula amb la mateixa estructura que la de Departaments que he anomenat Departaments2. A continuació he inserit un departament de vendes i posteriorment he tractat d'inserir un empleat el Departament del qual és aquest de la taula Departament2 i no un departament de la taula Departaments.

```
CREATE TABLE Departaments2 of Departament_T
(
    primary key (codi)
);
```

```
insert into Departaments2 values(Departament_T('C002','Vendes'));
```

Aquest ha estat el resultat de la inserció:

```
insert into Empleats values(Empleat_T('39688127R','Borja Vallejo',(select REF(P) from Departaments2 P where P.codi='C002')));
```

```
Error que empieza en la linea: 1 del comando :
insert into Empleats values(Empleat_T('39688127R','Borja Vallejo',(select REF(P) from Departaments2 P where P.codi='C002')))
Informe de error -
Error SQL: ORA-22889: REF value does not point to scoped table
22889. 00000 - "REF value does not point to scoped table"
*Cause:    An attempt was made to insert a REF value that does not point
          to the scoped table.
*Action:   Ensure that the REF values point to the scoped table.
```

Actualització dels valors referenciat

Primerament vaig a afegir un nou departament a la taula de Departaments i un nou empleat en aquest departament.

```
insert into Departaments values(Departament_T('C002','Vendes'));
insert into Empleats values(Empleat_T('39688127R','Borja Vallejo',(select REF(P) from Departaments P where P.codi='C002')));
```

Ara procedirem a canviar el codi del departament de Vendes que passarà a valdre V001.

```
update Departaments set codi='V001' where nom='Vendes';
```

Continuem fent una consulta de tots els empleats del departament de codi 'V001'.

```
select nif,nom from Empleats where DEREF(departament).codi='V001';
```

NIF	NOM
39688127R	Borja Vallejo

Acabem de mostrar un gran avantatge de l'utilització de referències respecte del model relacional pur. En el model relacional pur la instrucció de canvi del codi del departament de Vendes hagués fallat degut a una violació de la integritat referencial ja que deixariem en la taula d'empleats a empleats amb un codi de departament inexistent. A més en Oracle no tenim una clàusula `on update cascade` per a propagar canvis en cas d'actualitzar la clau primària en la taula referenciada.

En quant a la consulta que mostra els empleats serà àmpliament explicada a la següent secció.

Un altre aspecte a considerar és que ni tant sols necessitem la clàusula `foreign key` a la taula d'empleats per aconseguir integritat referencial. Mitjançant triggers es pot aconseguir fàcilment aquesta funcionalitat.

Considerem el següent supòsit: Si es tanca un departament els seus empleats no poden ser reubicats en un altre i per tant se'ls ha d'acomiadjar. Això vol dir que quan s'elimina un departament automàticament hem d'eliminar tots els seus empleats.

Amb la implementació que tenim és bastant fàcil aconseguir això només cal afegir una clàusula `on delete cascade` en el camp departament de la taula d'empleats.

De totes maneres anem a treure la clau forana de la taula empleats i implementar un trigger que automàticament elimini els empleats quan s'elimini el departament al qual pertanyen.

Primer hauríem d'eliminar la restricció de clau forana. Si l'haguéssim donat un nom a la restricció seria molt fàcil d'eliminar. La sintaxi per esborrar qualsevol tipus de restricció identificada per un nom seria aquesta:

```
ALTER TABLE table_name  
DROP CONSTRAINT constraint_name;
```

És recomanable donar un nom a les restriccions quan les creem. Si no es dóna un nom a una restricció automàticament l'Oracle l'assigna un nom.

Des l'entorn SQLDeveloper podem veure fàcilment els noms de les restriccions.

CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME
1 SYS_C007583	Primary_Key	(null)	(null)	(null)
2 SYS_C007584	Unique	(null)	(null)	(null)
3 SYS_C007587	Foreign_Key	(null)	PROVA	DEPARTAMENTOS

Aquí veiem (amb fons en color blau) que la restricció de clau forana es diu `SYS_C007587` i que està definida sobre el camp departament. Ara que sabem el nom de la restricció l'eliminem.

```
ALTER TABLE Empleats DROP CONSTRAINT SYS_C007587;
```

CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER
1 SYS_C007583	Primary_Key	(null)	(null)
2 SYS_C007584	Unique	(null)	(null)

Ja no apareix cap restricció de clau forana.

Com la restricció de clau forana implica la restricció d'àmbit, la eliminació de la clau forana ens deixa sense restricció d'àmbit per la qual l'he de definir (clàusula **SCOPE**).

La clàusula **SCOPE** s'utilitza per definir la restricció d'àmbit. Amb aquesta restricció indiquem que el camp amb la referència ha de contenir només una referència a un element que existeixi en una taula concreta. En el nostre cas les referències d'empleats a departaments han de ser a departaments que existeixin dintre de la taula Departaments.

Afegim la restricció d'àmbit modificant l'estructura de la taula amb la instrucció **ALTER TABLE**.

ALTER TABLE Empleats ADD SCOPE FOR (departament) **IS** Departaments;

La limitació de l'àmbit d'una referència es millor fer-la en el moment de la creació de la taula on aquesta està definida. En el nostre cas hauríem d'haver fet el següent:

```
CREATE TABLE Empleats of Empleat_T
(
    primary key (nif),
    scope for (departament) is Departaments
);
```

Creació del trigger que implementi l'acció on delete cascade

Veiem que el codi PLSQL per a la creació del trigger és força senzill.

```
CREATE OR REPLACE TRIGGER EmpleatsCascadeTRG
after delete on Departaments
begin
    delete from Empleats where departament is dangling;
end;
```

Anem ara a provar el funcionament eliminant el departament de Comptabilitat. Quan s'executa la instrucció que elimina el departament de Comptabilitat primer s'elimina aquest de la taula Departaments deixant els empleats de la taula Empleats amb el valor del camp departament en estat DANGLING. Tot seguit s'executa el trigger (clàusula AFTER del trigger) i elimina tots els empleats que tinguin el camp departament en estat DANGLING; és a dir els empleats que pertanyien al extint departament de Comptabilitat (Victòria i David).

La següent imatge mostra la situació abans d'eliminar el departament de Comptabilitat.

select nif,nom,DREF(departament).nom as departamento from Empleats;

NIF	NOM	DEPARTAMENT
39688127R	Borja Vallejo	Vendes
39672153B	Victòria Montalvo	Comptabilitat
21188594L	David Peñalver	Comptabilitat

DEPARTAMENTES

Columnas | Datos | Restricciones | Ordenar

CODI	NOM
1	V001 Vendes
2	C001 Comptabilitat

Després d'eliminar el departament de Comptabilitat la situació és la següent:

delete from Departaments where codi = 'C001';

select nif,nom,DREF(departament).nom as departamento from Empleats;

NIF	NOM	DEPARTAMENT
39688127R	Borja Vallejo	Vendes

DEPARTAMENTES

Columnas | Datos | Restricciones | Ordenar

CODI	NOM
1	V001 Vendes

Consulta de més d'una taula: amb joins o sense joins ?

Abans de res torno a inserir el departament de Comptabilitat i els dos empleats que havia esborrat abans.

```
insert into Departaments values(Departament_T('C001','Comptabilitat'));
insert into Empleats values(Empleat_T('39672153B','Victòria Montalvo',(select REF(P) from Departaments P where codi='C001')));
insert into Empleats values(Empleat_T('21188594L','David Peñalver',(select REF(P) from Departaments P where P.codi='C001')));
```

Farem ara un parell de consultes. La primera m'ha de dir per a cada empleat el departament on treballa. M'ha de mostrar el nom de l'empleat i el del departament.

Per fer aquesta consulta necessitem enllaçar les taules d'empleats i departaments (natural join). Amb les bases de dades relacionals pures això es feia igualant la clau primària de la taula referenciada amb la clau forana de la taula referenciadora. Si fem el mateix en aquest cas obtenint el següent resultat:

```
select E.nom,D.nom from Departaments D, Empleats E where D.codi=E.departament;
```

```
Error que empieza en la linea: 1 del comando -
select E.nom,D.nom from Departaments D, Empleats E where D.codi=E.departament
Error en la linea de comandos : 1 Columna : 65
Informe de error -
Error SQL: ORA-00932: inconsistent datatypes: expected NUMBER got REF PROVA.DEPARTAMENT_T
00932. 00000 - "inconsistent datatypes: expected %s got %s"
*Cause:
*Action:
```

El problema és que s'està tractant d'igualar una referència amb un valor numèric (el del camp codi de la taula de Departaments). Realment, el que hem d'igualar són les referències. La consulta correcta la faríem així:

```
select E.nom,D.nom from Departaments D, Empleats E where REF(D)=E.departament;
```

NOM	NOM
Borja Vallejo	Vendes
Victòria Montalvo	Comptabilitat
David Peñalver	Comptabilitat

Alternativament, es pot fer la mateixa consulta desreferenciant el camp departament dels empleats i igualant el codi de departament obtingut de la desreferència amb el de la taula de Departaments.

```
select E.nom,D.nom from Departaments D, Empleats E where D.codi=DEREF(E.departament).codi;
```

La següent consulta a fer em retornarà tots els empleats del departament de comptabilitat.

Altre cop estem en la mateixa situació d'abans; hem de fer un join de dues taules. La consulta seria aquesta:

```
select E.nif,E.nom from Departaments D, Empleats E where REF(D)=E.departament and D.nom='Comptabilitat';
```

NIF	NOM
39672153B	Victòria Montalvo
21188594L	David Peñalver

Amb aquestes dues consultes hem fet servir un join per enllaçar les dues taules. Però, és realment necessari fer un join de les taules d'empleats i departaments per resoldre les consultes que em demanaven ? La resposta és NO. Mitjançant la referència puc obtenir totes les dades del departament al qual pertany un empleat utilitzant l'operador DEREF.

La primera consulta sense joins la podríem fer així:

```
select nom,DEREF(departament).nom from Empleats;
```

NOM	DEREF(DEPARTAMENT).NOM
Borja Vallejo	Vendes
Victòria Montalvo	Comptabilitat
David Peñalver	Comptabilitat

La segona consulta seria aquesta:

```
select nif,nom from Empleats where DEREF(departament).nom='Comptabilitat';
```

NIF	NOM
39672153B	Victòria Montalvo
21188594L	David Peñalver

La conclusió és que quan s'utilitzen referències no cal fer joins entre taules, la pròpia referència és realment un join i l'únic que hem de fer per obtenir les dades és navegar per aquest enllaç per obtenir les dades de la taula referenciada.

De moment ens hem centrat molt en la utilització de sentències SQL. Més endavant, quan es tracti la implementació dels mètodes de les classes, utilitzarem molt més el llenguatge PL/SQL ja que aquests mètodes estaran escrits en aquest llenguatge.

Definició de mètodes de les classes

En una classe podem definir mètodes mitjançant la clàusula member. Els mètodes poden ser tant procediments com funcions i es poden utilitzar des d'SQL (per exemple en clàusules select) o en blocs i/o procediments de PL/SQL.

L'especificació de mètodes en una classe consta de dues fases. En la primera fase es defineix el mètode (la capçalera) amb els paràmetres que té i el tipus retornat.

En la segona fase implementem els mètodes en una secció especial de la classe destinada a tal propòsit i anomenada **type body**.

Per simplicitat anem a crear una nova classe anomenada Empleat2_T que tindrà els atributs nif, nom, sou i una referència a la classe Departament_T. A més definirem els mètodes getNif, getNom, getSou, getDepartament i apujarSou(increment number).

Fet això crearem una taula anomenada Empleats_2.

```
CREATE OR REPLACE TYPE Empleat2_T as OBJECT
(
    nif varchar2(9),
    nom varchar2(80),
    sou number(6),
    departament REF Departament_T,
    member function getNif return varchar2,
    member function getNom return varchar2,
    member function getSou return number,
    member function getDepartament return varchar2,
    member procedure apujarSou(increment number)
);
CREATE TABLE Empleats_2 of Empleat2_T
(
    primary key (nif),
    scope for (departament) is Departaments
);

CREATE OR REPLACE TYPE body Empleat2_T as
member function getNif return varchar2 as
begin
    return SELF.nif;
end getNif;

member function getNom return varchar2 as
begin
    return SELF.nom;
end getNom;

member function getSou return number as
begin
    return SELF.sou;
end getSou;

member function getDepartament return varchar2 as
dep varchar2(80);
begin
    select DEREF(departament).nom into dep from Empleats_2 E where E.nif=SELF.nif;
    return dep;
end getDepartament;

member procedure apujarSou(increment number) as
begin
    SELF.sou:=SELF.sou+increment;
end apujarSou;

end;
```

Els mètodes es defineixen com a member function o member procedure depenent si retornej o no algun valor. L'operador SELF fa referència a l'objecte que estem definint. És com l'operador this en java.

Inserirem unes quantes persones i farem algunes consultes.

```
insert into Empleats_2
values(Empleat2_T('39672153B','Víctoria Montalvo',2000,(select REF(P) from Departaments P where P.codi='C001')));

insert into Empleats_2
values(Empleat2_T('21188594L','David Peñalver',2500,(select REF(P) from Departaments P where P.codi='C001')));

insert into Empleats_2
values(Empleat2_T('39688127R','Borja Vallejo',1500,(select REF(P) from Departaments P where P.codi='V001')));
```

Exemple 1

Mostrar el nom, nif, sou i departament de tots els empleats (utilitzant els mètodes).

```
select E.getNif(),E.getNom(),E.getSou(),E.getDepartament() from Empleats_2 E;
```

E.GETNIF()	E.GETNOM()	E.GETSOU()	E.GETDEPARTAMENT()
39672153B	Víctoria Montalvo	2000	Comptabilitat
21188594L	David Peñalver	2500	Comptabilitat
39688127R	Borja Vallejo	1500	Vendes

Necessàriament hem d'utilitzar l'àlies de la taula (E). La següent consulta donaria error a l'hora d'executar-la.

```
select getNif(),getNom(),getSou(),getDepartament() from Empleats_2;
```

Exemple 2

Apujar 300 euros el sou de Borja Vallejo. Tracto de fer això:

```
select E.apujarSou(300) from Empleats_2 E where E.nom='Borja Vallejo';
```

Això és perquè no té sentit utilitzar els procediments en un select. Per tant per apujar el sou d'en Borja necessito fer un bloc (o procediment) PL/SQL. Aquest bloc de codi PL/SQL apuja el sou a l'empleat Borja Vallejo.

```
declare
borja Empleat2_T;
begin
    select value(E) into borja from Empleats_2 E where E.nom='Borja Vallejo';
    borja.apujarSou(300);
    update Empleats_2 E set E=borja where E.nom='Borja Vallejo';
end;
```

La següent captura mostra com efectivament l'hem apujat el sou a en Borja.

```
select E.getNif(),E.getNom(),E.getSou(),E.getDepartament() from Empleats_2 E;
```

E.GETNIF()	E.GETNOM()	E.GETSOU()	E.GETDEPARTAMENT()
39672153B	Víctoria Montalvo	2000	Comptabilitat
21188594L	David Peñalver	2500	Comptabilitat
39688127R	Borja Vallejo	1800	Vendes

Sobrecàrrega de mètodes. Sobrecàrrega del constructor i altres mètodes.

Oracle permet la sobrecàrrega de mètodes. El que és més habitual és la sobrecàrrega del constructor. Quan no es defineix cap constructor Oracle proporciona un per defecte. La crida al constructor es pot fer utilitzant l'operador new o sense utilitzar-lo (Oracle ja ho fa automàticament per nosaltres). Per defecte s'han de passar tants paràmetres com atributs tingui la classe.

Nosaltres podem sobreescrivir el constructor utilitzant la clàusula **constructor**. El constructor sempre s'ha definir amb el nom de la classe i la clàusula de retorn sempre ha de ser '[return SELF as RESULT](#)'. A la implementació en el type body s'ha de posar un '[return](#)' (sense res més).

Redefinirem la nostra classe Empleat2_T sobrecarregant el constructor de manera que es pugui instanciar opcionalment sense el sou i/o sense el departament de pertinença.

```

CREATE OR REPLACE TYPE Empleat2_T as OBJECT
( nif varchar2(9),
  nom varchar2(80),
  sou number(6),
  departament REF Departament_T,
  constructor function Empleat2_T(nif varchar2, nom varchar2) return SELF as RESULT,
  constructor function Empleat2_T(nif varchar2, nom varchar2, sou number) return SELF as RESULT,
  constructor function Empleat2_T(nif varchar2, nom varchar2, departament REF Departament_T) return SELF as RESULT,
  member function getNif return varchar2,
  member function getNom return varchar2,
  member function getSou return number,
  member function getDepartament return varchar2,
  member procedure apujarSou(increment number)
);
CREATE TABLE Empleats_2 of Empleat2_T
(
  primary key (nif),
  scope for (departament) is Departments
);
CREATE OR REPLACE TYPE body Empleat2_T as
constructor function Empleat2_T(nif varchar2, nom varchar2) return SELF as RESULT is
begin
  SELF.nif:=nif;
  SELF.nom:=nom;
  SELF.sou:=0;
  SELF.departament:=null;
  return;
end Empleat2_T;

constructor function Empleat2_T(nif varchar2, nom varchar2, sou number) return SELF as RESULT is
begin
  SELF.nif:=nif;
  SELF.nom:=nom;
  SELF.sou:=sou;
  SELF.departament:=null;
  return;
end Empleat2_T;

constructor function Empleat2_T(nif varchar2, nom varchar2, departament REF Departament_T) return SELF as RESULT is
begin
  SELF.nif:=nif;
  SELF.nom:=nom;
  SELF.sou:=0;
  SELF.departament:=departament;
  return;
end Empleat2_T;
member function getNif return varchar2 as
begin
  return SELF.nif;
end getNif;

member function getNom return varchar2 as
begin
  return SELF.nom;
end getNom;

member function getSou return number as
begin
  return SELF.sou;
end getSou;

member function getDepartament return varchar2 as
dep varchar2(80);
begin
  select DEREf(departament).nom into dep from Empleats_2 E where E.nif=SELF.nif;
  return dep;
end getDepartament;

member procedure apujarSou(increment number) as
begin
  SELF.sou:=SELF.sou+increment;
end apujarSou;
end;

```

Comprovem el funcionament del constructor sobrecarregat.

```
insert into Empleats_2
values(Empleat2_T('39672153B','Viktòria Montalvo',2000));

insert into Empleats_2
values(Empleat2_T('21188594L','David Peñalver',(select REF(P) from Departaments P where P.codi='C001')));

insert into Empleats_2
values(Empleat2_T('39688127R','Borja Vallejo'));

insert into Empleats_2
values(Empleat2_T('39771542R','Ramon Berenguer',3000,(select REF(P) from Departaments P where P.codi='V001')));

select E.getNif(),E.getNom(),E.getSou(),E.getDepartament() from Empleats_2 E;
```

Resultado de la Consulta			
SQL Todas las Filas Recuperadas: 4 en 0,008 segundos			
E.GETNIF()	E.GETNOM()	E.GETSOU()	E.GETDEPARTAMENT()
39672153B	Viktòria Montalvo	2000 (null)	
21188594L	David Peñalver	0 Comptabilitat	
39688127R	Borja Vallejo	0 (null)	
39771542R	Ramon Berenguer	3000 Vendes	

La sobrecàrrega de mètodes es fa de manera anàloga a com ho fem amb el constructor. Per exemple podríem haver sobrecarregat el mètode apujarSou de manera que si no li passem cap paràmetre apuji el sou en 100 euros. Per fer això hauríem de:

Afegeix la definició del mètode a la definició de la classe.	Afegeix la implementació al type body.
member procedure apujarSou	member procedure apujarSou as begin SELF.sou:=SELF.sou+100; end apujarSou;

Comprovació: Apugem el sou a Victòria en 100 euros (opcio per defecte) i apugem el sou a David en 800 euros.

```
declare
empl Empleat2_T;
begin
select value(E) into empl from Empleats_2 E where E.nom='Viktòria Montalvo';
empl.apujarSou();
update Empleats_2 E set E=empl where E.nom='Viktòria Montalvo';
select value(E) into empl from Empleats_2 E where E.nom='David Peñalver';
empl.apujarSou(800);
update Empleats_2 E set E=empl where E.nom='David Peñalver';
end;
```

Abans				Després			
E.GETNIF()	E.GETNOM()	E.GETSOU()	E.GETDEPARTAMENT()	E.GETNIF()	E.GETNOM()	E.GETSOU()	E.GETDEPARTAMENT()
39672153B	Viktòria Montalvo	2000 (null)		39672153B	Viktòria Montalvo	2100 (null)	
21188594L	David Peñalver	0 Comptabilitat		21188594L	David Peñalver	800 Comptabilitat	

Comparació entre objectes amb la clàusula MAP.

Per comparar objectes utilitzarem les clàusules map i order. Quan definim un mètode amb la clàusula map nosaltres podem comparar dos objectes amb els operadors relacionals com si estiguéssim comparant valors numèrics. Només pot haver un mètode amb la clàusula map i aquest ha de retornar un valor d'un tipus que pugui ser comparable o utilitzat en ordenacions. Els mètodes map ens permeten comparar objecte (per exemple podem fer obj1 > obj2) i es criden automàticament en les clàusules DISTINCT, GROUP BY i ORDER BY. Anem a afegir un atribut anomenat data_naixament de tipus date i farem un mètode map que retornarà l'edat i ens permetrà saber quins empleats són més grans o petits que altres.

```

CREATE OR REPLACE TYPE Empleat2_T AS OBJECT
( nif VARCHAR2(9),
  nom VARCHAR2(80),
  data_naixament DATE,
  sou NUMBER(6),
  departament REF Departament_T,
  CONSTRUCTOR FUNCTION Empleat2_T(nif VARCHAR2, nom VARCHAR2, data_naixament DATE) RETURN SELF AS RESULT,
  CONSTRUCTOR FUNCTION Empleat2_T(nif VARCHAR2, nom VARCHAR2, data_naixament DATE, sou NUMBER) RETURN SELF AS RESULT,
  CONSTRUCTOR FUNCTION Empleat2_T(nif VARCHAR2, nom VARCHAR2, data_naixament DATE, departament REF Departament_T) RETURN SELF AS RESULT,
  MAP MEMBER FUNCTION edat RETURN NUMBER,
  MEMBER FUNCTION getNif RETURN VARCHAR2,
  MEMBER FUNCTION getNom RETURN VARCHAR2,
  MEMBER FUNCTION getSou RETURN NUMBER,
  MEMBER FUNCTION getDepartament RETURN VARCHAR2,
  MEMBER PROCEDURE apujarSou,
  MEMBER PROCEDURE apujarSou(INCREMENT NUMBER)
);
CREATE TABLE Empleats_2 OF Empleat2_T
(
  PRIMARY KEY (nif),
  SCOPE FOR (departament) IS Departaments
);
CREATE OR REPLACE TYPE BODY Empleat2_T AS
CONSTRUCTOR FUNCTION Empleat2_T(nif VARCHAR2, nom VARCHAR2, data_naixament DATE) RETURN SELF AS RESULT IS
BEGIN
  SELF.nif:=nif;
  SELF.nom:=nom;
  SELF.data_naixament :=data_naixament;
  SELF.sou:=0;
  SELF.departament:=NULL;
  RETURN;
END Empleat2_T;

CONSTRUCTOR FUNCTION Empleat2_T(nif VARCHAR2, nom VARCHAR2, data_naixament DATE, sou NUMBER) RETURN SELF AS RESULT IS
BEGIN
  SELF.nif:=nif;
  SELF.nom:=nom;
  SELF.data_naixament :=data_naixament;
  SELF.sou:=sou;
  SELF.departament:=NULL;
  RETURN;
END Empleat2_T;

CONSTRUCTOR FUNCTION Empleat2_T(nif VARCHAR2, nom VARCHAR2, data_naixament DATE, departament REF Departament_T) RETURN SELF AS RESULT IS
BEGIN
  SELF.nif:=nif;
  SELF.nom:=nom;
  SELF.data_naixament :=data_naixament;
  SELF.sou:=0;
  SELF.departament:=departament;
  RETURN;
END Empleat2_T;

MAP MEMBER FUNCTION edat RETURN NUMBER AS
BEGIN
  RETURN (SYSDATE - SELF.data_naixament)/365;
END edat;

MEMBER FUNCTION getNif RETURN VARCHAR2 AS
BEGIN
  RETURN SELF.nif;
END getNif;

MEMBER FUNCTION getNom RETURN VARCHAR2 AS
BEGIN
  RETURN SELF.nom;
END getNom;

MEMBER FUNCTION getSou RETURN NUMBER AS
BEGIN
  RETURN SELF.sou;
END getSou;

```

```

member function getDepartament return varchar2 as
dep varchar2(80);
begin
  select DEREF(departament).nom into dep from Empleats_2 E where E.nif=SELF.nif;
  return dep;
end getDepartament;

member procedure apujarSou(increment number) as
begin
  SELF.sou:=SELF.sou+increment;
end apujarSou;

member procedure apujarSou as
begin
  SELF.sou:=SELF.sou+100;
end apujarSou;

end;

```

```

insert into Empleats_2
values(Empleat2_T('39672153B','Viktòria Montalvo',to_date('27/03/1971','dd/mm/yyyy'),2000));

```

```

insert into Empleats_2
values(Empleat2_T('21188594L','David Peñalver',to_date('12/10/1961','dd/mm/yyyy'),
(select REF(P) from Departaments P where P.codí='C001')));

```

```

insert into Empleats_2
values(Empleat2_T('39688127R','Borja Vallejo',to_date('03/07/1985','dd/mm/yyyy')));

```

```

insert into Empleats_2
values(Empleat2_T('39771542R','Ramon Berenguer',3000,to_date('30/01/1974','dd/mm/yyyy'),
(select REF(P) from Departaments P where P.codí='V001')));

```

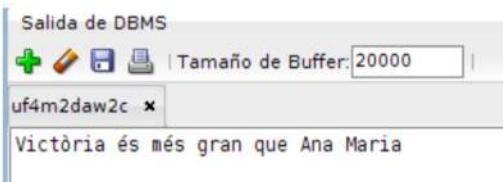
NIF	NOM	SOU	DEPARTAMENT	DATA_NAIXAMENT
39672153B	Viktòria Montalvo	2000 (null)		27/03/71
21188594L	David Peñalver	0 [UF4M2DAW2C.DEPARTAMENT_T]		12/10/61
39688127R	Borja Vallejo	0 (null)		03/07/85
39771542R	Ramon Berenguer	3000 [UF4M2DAW2C.DEPARTAMENT_T]		30/01/74

Comprovem el funcionament del mètode map en el següent bloc de PL/SQL.

```

declare
empl Empleat2_T;
empl2 Empleat2_T;
begin
  select value(E) into empl from Empleats_2 E where E.nom='Viktòria Montalvo';
  empl2:= new Empleat2_T('39555123W','Ana Maria López',to_date('22/06/1989','dd/mm/yyyy'));
  if (empl > empl2) then
    DBMS_OUTPUT.PUT_LINE('Viktòria és més gran que Ana Maria');
  elsif (empl < empl2) then
    DBMS_OUTPUT.PUT_LINE('Viktòria és més petita que Ana Maria');
  else
    DBMS_OUTPUT.PUT_LINE('Viktòria i Ana Maria tenen la mateixa edat');
  end if;
end;

```



Consulta que retorna tots els empleats que siguin més grans que un donat.

```
select E.getNif(), E.getNom(), E.getSou(), to_char(E.data_naixament,'dd/mm/yyyy') as data_n from Empleats_2 E where value(E) > Empleat2_T('39331257X','Verónica Martínez',to_date('05/04/1972','dd/mm/yyyy'));
```

E.GETNIF()	E.GETNOM()	E.GETSOU()	DATA_N
39672153B	Victòria Montalvo	2000	27/03/1971
21188594L	David Peñalver	900	12/10/1961

Consulta que retorna tots els empleats menors que Victòria Montalvo.

```
select E.getNif(), E.getNom(), E.getSou(), to_char(E.data_naixament,'dd/mm/yyyy') as data_n from Empleats_2 E where value(E) < (select value(P) from Empleats_2 P where P.getNif()=39672153B);
```

E.GETNIF()	E.GETNOM()	E.GETSOU()	DATA_N
39688127R	Borja Vallejo	1500	03/07/1985
39771542R	Ramon Berenguer	3000	30/01/1974

Comparació d'objectes amb la clàusula ORDER.

Amb la clàusula order nosaltres no podem establir un ordre entre els objectes contràriament al que passa amb la clàusula MAP. Amb la clàusula ORDER només podem decidir si un objecte és menor, major o igual que un altre. La sintaxi d'un mètode ORDER és més estricta que la dels mètodes MAP. Un mètode ORDER té un paràmetre que és de la mateixa classe que la classe en la qual estem definint aquest mètode. A més ha de retornar un valor enter (INTEGER) que necessàriament ha de valer 1, -1 ó 0 si l'objecte és major, menor o igual respectivament que l'objecte passat com a paràmetre.

Només es pot utilitzar un mètode MAP o ORDER però no els dos simultàniament.

Exemple

A continuació definirem la classe Complexe_T que representarà un número complex amb les operacions partReal, partImaginaria, obtenirModul, obtenirConjugat, sumaComplexe, producteComplexe i el constructor sobrecarregat. A més, fet això crearem un orden mètode anomenat compararComplexes que ens permetrà comparar números complexes.

Especificació de la classe Complexe_T.

```
create or replace type Complexe_T as object
/* TODO enter attribute and method declarations here */
    preal number(10,5),
    pimaginaria number(10,5),
    constructor function Complexe_T return SELF as RESULT,
    constructor function Complexe_T(preal number) return SELF as RESULT,
    member function partReal return number,
    member function partImaginaria return number,
    member function obtenirConjugat return Complexe_T,
    member function sumaComplexe(c1 Complexe_T) return Complexe_T,
    member function producteComplexe(c1 Complexe_T) return Complexe_T,
    member function obtenirModul return number,
    member procedure mostrarComplexe,
    order member function compararComplexes(c1 Complexe_T) return INTEGER
);
```

Implementació dels mètodes definits a la classe Complexe_T.

```

create or replace type body Complexe_T as
constructor function Complexe_T return SELF as RESULT as
begin
  SELF.preal:=0;
  SELF.pimaginaria:=0;
  return;
end Complexe_T;

constructor function complexe_t(preal number) return SELF as RESULT as
begin
  SELF.preal:=preal;
  SELF.pimaginaria:=0;
  return;
end Complexe_T;

member function partReal return number as
begin
  return SELF.preal;
end partReal;

member function partImaginaria return number as
begin
  return SELF.pimaginaria;
end partImaginaria;

member function obtenerConjugat return Complexe_T as
begin
  return Complexe_T(SELF.preal,(-1)*SELF.pimaginaria);
end obtenerConjugat;

member function sumaComplexe(c1 Complexe_T) return Complexe_T as
begin
  return Complexe_T(SELF.preal + c1.preal,SELF.pimaginaria + c1.pimaginaria);
end sumaComplexe;

member function producteComplexe(c1 Complexe_T) return Complexe_T as
pr number(10,5);
pi number(10,5);
begin
  pr:=SELF.preal * c1.preal - SELF.pimaginaria * c1.pimaginaria;
  pi:=SELF.pimaginaria * c1.preal + SELF.preal * c1.pimaginaria;
  return Complexe_T(pr,pi);
end producteComplexe;

member function obtenerModul return number as
modul number(10,5);
begin
  modul:=SQRT(SELF.preal * SELF.preal + SELF.pimaginaria * SELF.pimaginaria);
  return modul;
end obtenerModul;

member procedure mostrarComplexe as
begin
  if (SELF.pimaginaria < 0) then
    DBMS_OUTPUT.PUT_LINE(SELF.preal || ' - ' || ABS(SELF.pimaginaria) || 'i');
  else
    DBMS_OUTPUT.PUT_LINE(SELF.preal || ' + ' || SELF.pimaginaria || 'i');
  end if;
end;

order member function compararComplexes(c1 complexe_t) return integer as
ret integer;
begin
  ret:=0;
  if (SELF.obtenerModul() < c1.obtenerModul()) then
    ret:=-1;
  elsif (SELF.obtenerModul() > c1.obtenerModul()) then
    ret:=1;
  end if;
  return ret;
end compararComplexes;

end;

```

Bloc de codi PL/SQL.

```

declare
c1 Complexe_T;
c2 Complexe_T;
c3 Complexe_T;
begin
  c1:= new Complexe_T(1,2);
  c2:= new Complexe_T(1,1);
  DBMS_OUTPUT.PUT('c1 = ');
  c1.mostrarComplexe();
  DBMS_OUTPUT.PUT('c2 = ');
  c2.mostrarComplexe();
  c3:=c1.obtenirConjugat();
  DBMS_OUTPUT.PUT('Conjugat de c1 = ');
  c3.mostrarComplexe();
  c3:=c1.sumaComplexe(c2);
  DBMS_OUTPUT.PUT('Suma de c1 i c2 = ');
  c3.mostrarComplexe();
  c3:=c1.producteComplexe(c2);
  DBMS_OUTPUT.PUT('Producte de c1 i c2 = ');
  c3.mostrarComplexe();
  DBMS_OUTPUT.PUT_LINE('Mòdul c1: ' || c1.obtenirModul());
  DBMS_OUTPUT.PUT_LINE('Mòdul c2: ' || c2.obtenirModul());
  if (c1 > c2) then
    DBMS_OUTPUT.PUT_LINE('c1 > c2');
  elsif (c1 < c2) then
    DBMS_OUTPUT.PUT_LINE('c1 < c2');
  else
    DBMS_OUTPUT.PUT_LINE('c1 = c2');
  end if;
end;

```

L'execució del bloc de codi PL/SQL mostra, entre altres coses, com funciona el mètode definit amb la clàusula ORDER. Els complexos de major mòdul són més grans que els que tenen menor mòdul.

Col·leccions

Un tipus col·lecció és un tipus de dades (una classe) que ens permet emmagatzemar un conjunt d'elements del mateix tipus. El tipus base d'una col·lecció pot ser un tipus elemental o bé un tipus objecte definit per l'usuari, fins i tot pot ser un altre tipus col·lecció. Oracle té tres tipus de col·lecció: arrays, arrays associatius i taules anidades. La següent taula mostra les característiques de cada tipus en Oracle.

Table 5–1 PL/SQL Collection Types

Collection Type	Number of Elements	Index Type	Dense or Sparse	Uninitialized Status	Where Defined	Arrays
Associative array (or index-by table)	Unspecified	String or PLS_INTEGER	Either	Empty	In PL/SQL block or package	En Oracle els arrays defineixen com de tipus VARRAY . Un VARRAY és un conjunt ordenat d'elements on els elements són accessibles per posició (vaja com els arrays típics de qualsevol llenguatge de programació).
VARRAY (variable-size array)	Specified	Integer	Always dense	Null	In PL/SQL block or package or at schema level	
Nested table	Unspecified	Integer	Starts dense, can become sparse	Null	In PL/SQL block or package or at schema level	

Quan es declaren se'ls ha de donar una mida màxima encara que després es poden expandir (VARRAY vol dir array de mida variable).

Sintaxi VARRAY

```
TYPE type_name IS {VARRAY | VARYING ARRAY} (size_limit)
OF element_type [NOT NULL];
```

Exemple

Volem emmagatzemar informació adicional sobre els telèfons dels empleats. Per donar resposta a aquest nou requeriment podem utilitzar un varray pels telèfons (podem suposar que cada empleat no pot tenir més de 10 telèfons). Per tant primer definirem un nou tipus que representi el vector de telèfons i definirem un nou atribut en la classe Empleat_T del tipus definit anteriorment.

Com la classe Empleat_T existeix haurem de modificar l'estrucció de la classe per afegir el nou atribut. Això ho fem amb la instrucció alter type. L'opció cascade és per difondre els canvis als elements de la base de dades que depenen d'aquest tipus.

```
CREATE TYPE telefons_T is VARRAY(10) of varchar2(10);
```

```
ALTER TYPE Empleat_T add attribute (telefons telefons_T) cascade;
```

Hauria estat definir l'atribut en el moment de la creació del tipus.

```
CREATE OR REPLACE TYPE Empleat_T as OBJECT
```

```
( 
    nif varchar2(9),
    nom varchar2(80),
    departament REF Departament_T,
    telefons telefons_T
);
```

Inserció de dades: exemple

Afegir un nou empleat amb 3 telèfons (no l'assiganaré a cap departament).

```
insert into Empleats values(Empleat_T('25145278Z','Valeria Bermejo',null,telefons_T('973112233','965213267','659171119')));
```

Com els tipus col·lecció són classes s'instancien com qualsevol classe. En aquest cas el nom de la classe, telefons_T, seguit de la tirallonga de telèfons.

Els tipus varray s'emmagatzemen com una columna més de la taula, però si el valor supera els 4000 bytes aleshores s'emmagatzema a banda com un tipus BLOB.

Arrays associatius

Els arrays associatius són estructures de dades que permeten emmagatzemar un conjunt no ordenat de valors. Aquests arrays no tenen una mida fixa i creixen dinàmicament quan es van inserir elements. Cada element està format per una parella on un component és diu clau i l'altre és diu valor. Les claus són identifiquen cada element a l'array i per tant no poden estar repetides. L'accés es fa per clau i es retorna el valor associat a la clau (per això se'n diu array associatiu; donada una clau retorna el valor associat a aquesta clau). Aquest tipus de dades no poden ser utilitzades com a tipus de camps de taules ni es pot definir un tipus array associatiu que formi part de l'esquema de la base de dades. Només els podrem utilitzar en blocs i/o procediment i funcions de PL/SQL per la qual cosa es diu que es un tipus de dades no persistent ja que quan s'acaba d'executar el bloc de codi PL/SQL que els contenen es destrueixen.

Exemple

Definirem un array associatiu amb la població de diversos pobles de la província de Tarragona. Aquest exemple de codi PL/SQL mostra com es defineix un array associatiu i diferents operacions que es poden fer amb ells.

```
declare
/* Definició de l'array associatiu de la població. La clau és de tipus varchar2(80) i es corresponent amb el nom de la població.
   El valor és un numèric i es correspon amb la quantitat d'habitants.*/
type Poblacio_T is table of number index by varchar2(80);

/* Variable per a l'array i l'index */
poblacio Poblacio_T;
poble varchar2(80);

begin
  /* Afegir elements a l'array associatiu població */
  poblacio('Tarragona') := 134163;
  poblacio('Salou') := 23398;
  poblacio('Cambrils') := 29112;
  poblacio('Reus') := 104835;
  poblacio('El Vendrell') := 33340;
  poblacio('Calafell') := 21871;
  poblacio('Deltebre') := 11063;

  /* Mostra els els elements de l'array utilitzant el mètode NEXT dels arrays associatius */
  poble := poblacio.FIRST;
  while poble is not null loop
    DBMS_Output.PUT_LINE (poble || ':' || poblacio(poble) || ' habitants');
    poble := poblacio.NEXT(poble);
  end loop;
end;
```

El resultat de l'execució és:

```
Salida de DBMS
+ | Tamaño de Buffer: 20000 |
uf4m2daw2c x
Calafell : 21871 habitants
Cambrils : 29112 habitants
Deltebre : 11063 habitants
El Vendrell : 33340 habitants
Reus : 104835 habitants
Salou : 23398 habitants
Tarragona : 134163 habitants
```

Taules anidades

En Oracle una taula anidada es diu NESTED TABLE i és un tipus de dades que ens permet tenir un conjunt no

afitat d'elements del mateix tipus base. Les taules anidades poden ser definides tant en blocs de codi PL/SQL com en la base de dades com objectes que formen part de l'esquema de la base de dades com per exemple camps de taules. Per aquesta raó es diu que aquest tipus de dades és persistent.

Quan es defineixen com a camps d'una taula aleshores no és crea una taula per a cada filera de la taula pare sinó que s'emmagatzem tots en una única taula. Oracle gestiona internament i de manera transparent per a l'usuari la distribució de les dades en les taules anidades i sap a quina filera de la taula pare pertany cada registre de la taula anidada.

Sintaxi de declaració d'una taula anidada en PL/SQL

```
DECLARE  
TYPE type_name IS TABLE OF element_type [NOT NULL] ;
```

Sintaxi de declaració d'una taula anidada en l'esquema de la base de dades.

```
CREATE [OR REPLACE] TYPE type_name IS TABLE OF [element_type] [NOT  
NULL] ;
```

Exemple taules anidades

Imaginem que ara necessitem tenir a la nostra taula d'empleats un camp que sigui les aficions dels empleats. Supossem també que hem optat per emmagatzemar les aficions en una taula anidada. Com canviaria la definició del tipus (classe) Empleat_T?

Primer definiríem la taula anidada de la següent manera.

```
CREATE OR REPLACE TYPE hobbies_T as TABLE of varchar2(80);
```

A continuació canviaríem la definició de la classe Empleat_T per a que tingui la taula d'aficions.

```
ALTER TYPE Empleat_T add attribute (aficions hobbies_T cascade);
```

```
ALTER TABLE Empleats upgrade nested table aficions store as taula_aficions; (A partir de la versió 11g)
```

Realment el que s'hauria d'haver fet és crear el tipus Empleat_T amb aquest camp des del començament i després crear la taula d'empleats. La definició del tipus Empleat_T i de la taula Empleats hagués estat aquesta (tornaré a refer la taula i inseriré de nou les dades):

```
CREATE OR REPLACE TYPE Empleat_T as OBJECT  
(  
    nif varchar2(9),  
    nom varchar2(80),  
    departament REF Departament_T,  
    telefons telefons_T,  
    aficions hobbies_T  
)  
  
CREATE TABLE Empleats of Empleat_T  
(  
    primary key (nif),  
    foreign key (departament) references Departaments  
) nested table aficions store as taula_aficions;
```

Operacions sobre els tipus col·lecció VARRAY i NESTED TABLE

Ex1. A continuació insereixo 3 registres d'empleats on es veu com s'insereix una referència al departament de pertinença, els telèfons que són de tipus varray i les aficions que són de tipus nested table.

```
insert into Empleats  
values(Empleat_T('39688127R','Borja Vallejo',null,  
telefons_T('659637215','976412345','977212845'),hobbies_T('Tenis taula','Natació','Boxa','Futbol')));
```

```
insert into Empleats  
values(Empleat_T('39672153B','Viktòria Montalvo',(select REF(P) from Departaments P where P.codi='C001'),  
telefons_T('965121234','659312487'),hobbies_T('Paddle')));
```

```
insert into Empleats
values(Empleat_T('21188594L','David Peñalver',(select REF(P) from Departaments P where P.codi='C001'),
telefons_T('958252177'),hobbies_T('Paddle Surf','Escacs','Cinema','Teatre')));
```

Ex 2. Quan mostrem els empleats consultant la taula des de la perspectiva del model relacional no surten les dades de tipus col·lecció.

select * from Empleats;

NIF	NOM	DEPARTAMENT
39688127R	Borja Vallejo	UF4M2DAW2C.DEPARTAMENT_T('C001','Comptabilitat')
39672153B	Victòria Montalvo	UF4M2DAW2C.DEPARTAMENT_T('C001','Comptabilitat')
21188594L	David Peñalver	

Ex 3. Quan fem la consulta utilitzant l'operador value aleshores sí que surten els tipus col·lecció però no surt el valor referenciat del departament.

select value(P) from Empleats P;

VALUE(P)
UF4M2DAW2C.EMPLEAT_T('39688127R','Borja Vallejo',NULL,UF4M2DAW2C.TELEFONS_T(659637215,976412345,977212845),UF4M2DAW2C.HOBIES_T(Tenis taula,Natació,Boxa,Futbol))
UF4M2DAW2C.EMPLEAT_T('39672153B','Victòria Montalvo','oracle.sql.REF@324ea55f',UF4M2DAW2C.TELEFONS_T(965121234,659312487),UF4M2DAW2C.HOBIES_T(Paddle Surf))
UF4M2DAW2C.EMPLEAT_T('21188594L','David Peñalver','oracle.sql.REF@324ea55f',UF4M2DAW2C.TELEFONS_T(958252177),UF4M2DAW2C.HOBIES_T(Paddle Surf,Escacs,Cinema,Teatre))

Ex 4. Mostrar les aficions de tots els empleats.

select P.nom, P.aficions from Empleats P;

NOM	AFICIONS	Ex 5. Mostrar totes les aficions de Borja Vallejo
Borja Vallejo	UF4M2DAW2C.HOBIES_T('Tennis taula','Natació','Boxa','Futbol')	
Victòria Montalvo	UF4M2DAW2C.HOBIES_T('Paddle Surf')	
David Peñalver	UF4M2DAW2C.HOBIES_T('Paddle Surf','Escacs','Cinema','Teatre')	

(una per línia).

select NT.column_value as Aficions from Empleats E, TABLE (E.aficions) NT where lower(E.nom)='borja vallejo';

AFICIONS
Tennis taula
Natació
Boxa
Futbol

L'operador TABLE s'utilitza per accedir a les dades d'una taula anidada. Internament es fa un join de la taula pare (Empleats en el nostre cas) i la taula anidada (taula_aficions).

Ex 6. Mostrar tots els telèfons dels empleats que tenen assignar algun departament. Mostra també el nom del departament.

select P.nom as Nom, DEREF(P.departament).nom as Departament,P.telefons from Empleats P where P.departament is not null;

NOM	DEPARTAMENT	TELEFONS
Victòria Montalvo	Comptabilitat	UF4M2DAW2C.TELEFONS_T('965121234','659312487')
David Peñalver	Comptabilitat	UF4M2DAW2C.TELEFONS_T('958252177')

Ex 7. Mostrar tots els telèfons de l'empleat Borja Vallejo.

select NT.column_value as Telèfons from Empleats E, TABLE (E.telefons) NT where E.nom='Borja Vallejo';

TELÈFONS
659637215
976412345
977212845

Veiem que les consultes sobre les dades de tipus varray es fan de la mateixa manera que es fan sobre les dades de tipus nested array.

Ex 8. Afegir una afició a Borja Vallejo.

```
insert into table (select aficions from Empleats where nom='Borja Vallejo') values('Submarinisme');
```

```
select NT.column_value as Aficions from Empleats E, TABLE (E.aficions) NT where E.nom='Borja Vallejo';
AFICIONS
-----
Submarinisme
Tenis taula
Natació
Boxa
Futbol
```

```
insert into the (select aficions from Empleats where nom like '%Montalvo') values('Jocs de rol');
```

```
select aficions from Empleats where nom like '%Montalvo';
AFICIONS
-----
UF4M2DAW2C.HOBBIEST('Jocs de rol','Paddle')
```

Ara inseriré un altre empleat sense hobbies ni telèfons.

```
insert into Empleats
values(Empleat_T('39671221X','Josep Maria Moncusí',(select REF(P) from Departaments P where P.codig='V001'),null,null));
```

Ex 9. Afegir una afició a Josep Maria

```
insert into table (select aficions from Empleats where nom='Josep Maria Moncusí') values('Viatjar');
```

```
Error que empieza en la linea: 1 del comando :
insert into table (select aficions from Empleats where nom='Josep Maria Moncusí') values('Viatjar')
Informe de error -
Error SQL: ORA-22908: referencia a valor de tabla NULL
22908. 00000 - "reference to NULL table value"
*Cause:   The evaluation of the THE subquery or nested table column
          resulted in a NULL value implying a NULL table instance.
          The THE subquery or nested table column must identify a
          single non-NULL table instance.
*Action:  Ensure that the evaluation of the THE subquery or nested table
          column results in a single non-null table instance. If happening
          in the context of an insert statement where the THE subquery is
          the target of an insert, then ensure that an empty nested table
          instance is created by updating the nested table column of the
          parent table's row specifying an empty nested table constructor.
```

El missatge d'error explica la situació i dóna la solució bastant clarament. Hem de fer un update cridant al constructor de la classe de la taula anidada.

```
update Empleats set aficions = hobbies_T('Viajar') where nom='Josep Maria Moncusí';
```

No hi havia cap afició introduïda. A partir d'ara les aficions s'introduixen amb un insert.

```
insert into table (select aficions from Empleats where nom='Josep Maria Moncusí') values('Cuina');
```

NOM	AFICIONS
Josep Maria Moncusí	UF4M2DAW2C.HOBBIEST('Viajar','Cuina')
Borja Vallejo	UF4M2DAW2C.HOBBIEST('Submarinisme','Tenis taula','Natació','Boxa','Futbol')
Victòria Montalvo	UF4M2DAW2C.HOBBIEST('Jocs de rol','Paddle')
David Peñalver	UF4M2DAW2C.HOBBIEST('Paddle Surf','Escacs','Cinema','Teatre')

Ex 10. Canviar l'afició Viajar de Josep Maria per Viatjar.

```
update table (select aficions from Empleats where nom='Josep Maria Moncusí') AF
set AF.column_value='Viatjar' where AF.column_value='Viajar';
```

Ex 11. Mostrar totes les aficions dels empleats del departament de codi 'V001'.

```
select nom,aficions from where Empleats DEREf(departament).codi='V001';
```

NOM	AFICIONS		
Josep Maria Moncusí	UF4M2DAW2C.HOBBIEST('Viatjar', 'Navegar')	Ex 11. Treure l'afició de	

Navegar a Josep Maria Moncusí.

```
delete from table (select aficions from Empleats where nom='Josep Maria Moncusí') AF where AF.column_value='Navegar';
```

Abans de l'operació	Després de l'operació		
NOM	AFICIONS	NOM	AFICIONS
Josep Maria Moncusí	UF4M2DAW2C.HOBBIEST('Viatjar', 'Navegar')	Josep Maria Moncusí	UF4M2DAW2C.HOBBIEST('Viatjar')

Els operadors table i the són intercanviables; fan el mateix.

Ex 12. Afegir un telefon a David Peñalver.

```
insert into table (select telefons from Empleats where nom='David Peñalver') values('661253718');
```

Sobre les col·leccions de tipus VARRAY no es podem fer operacions d'inserció, actualització o eliminació amb SQL; necessitem PL/SQL per fer això.

Mètodes de les classes col·lecció (varray i nested table)

La següent taula mostra els diferents mètodes que tenen els tipus col·lecció.

La sintaxi general és <nom col·lecció>.mètode (Exemple: telefons.FIRST on telefons és un varray).

EXISTS(<index>)	Em diu si existe un element en la col·lecció. Retorna true o false.
COUNT	Retorna la quantitat d'elements introduïts en una col·lecció.
LIMIT	Retorna la capacitat de la col·lecció (nombre màxim d'elements).
FIRST	Retorna el primer element de la col·lecció (està a la posició 1 no la 0).
LAST	Retorna el darrer element de la col·lecció.
PRIOR(<index>)	Retorna l'element anterior al de l'index passat com a paràmetre.
NEXT(<index>)	Retorna l'element posterior al de l'index passat com a paràmetre.
EXTEND[(<valor>)]	Afegeix un (o més d'un) valors nuls ampliant la capacitat de la col·lecció.
TRIM[(<valor>)]	Elimina un (o més d'un) valors des de la darrera posició cap a la primera.
DELETE[(n,m)]	Elimina tots els elements o l'element indicat o un rang d'elements entre n i m.

Ex 12b. Afegir un telefon a David Peñalver (amb PL/SQL).

```
create or replace procedure inserirtelefon(nif_empl in varchar2,telefon in varchar2) as
taula_af telefons_T;
num number(10);

begin
select telefons into taula_af from Empleats where nif=nif_empl;
num:=taula_af.COUNT;
taula_af.EXTEND;
taula_af(num+1):=telefon;
update Empleats set telefons=taula_af where nif=nif_empl;
end inserirtelefon;

execute inserirtelefon('21188594L','661253718');
```

Hem fet un procediment per inserir un telèfon a la taula d'empleats donat el nif de l'empleat i el telèfon a afegir. La següent taula mostra les dades abans i després d'executar el procediment PL/SQL.

PL/SQL és molt més versàtil que l'SQL i ens permet fer moltes més operacions sobre les dades de la base de dades. Podem observar també els mètodes COUNT i EXTEND de les col·leccions.

Abans de l'operació	Després de l'operació								
<table><thead><tr><th>NOM</th><th>TELEFONS</th></tr></thead><tbody><tr><td>David Peñalver</td><td>UF4M2DAW2C.TELEFONS_T('958252177')</td></tr></tbody></table>	NOM	TELEFONS	David Peñalver	UF4M2DAW2C.TELEFONS_T('958252177')	<table><thead><tr><th>NOM</th><th>TELEFONS</th></tr></thead><tbody><tr><td>David Peñalver</td><td>UF4M2DAW2C.TELEFONS_T('958252177', '661253718')</td></tr></tbody></table>	NOM	TELEFONS	David Peñalver	UF4M2DAW2C.TELEFONS_T('958252177', '661253718')
NOM	TELEFONS								
David Peñalver	UF4M2DAW2C.TELEFONS_T('958252177')								
NOM	TELEFONS								
David Peñalver	UF4M2DAW2C.TELEFONS_T('958252177', '661253718')								

Ex 13. Modificar un telefon de David Peñalver (amb PL/SQL).

```
create or replace procedure modificartelefon(nif_empl in varchar2, telefonold in varchar2, telefonnew in varchar2) as
taula_af telefons_T;
i number(5);
fi boolean;

begin
select telefons into taula_af from Empleats where nif=nif_empl;
i:=1;
fi:=false;
while (i<=taula_af.COUNT and not fi) loop
  if (taula_af(i) = telefonold) then
    taula_af(i):=telefonnew;
    fi:=true;
  end if;
  i:=i+1;
end loop;
update Empleats set telefons=taula_af where nif=nif_empl;
end modificartelefon;

execute modificartelefon('21188594L','958252177','958252176');
```

Hem fet un procediment per modificar un telèfon a la taula d'empleats donat el nif de l'empleat, el telèfon a modificar i el nou valor del telèfon. La següent taula mostra les dades abans i després d'executar el procediment PL/SQL.

Abans de l'operació	Després de l'operació								
<table><thead><tr><th>NOM</th><th>TELEFONS</th></tr></thead><tbody><tr><td>David Peñalver</td><td>UF4M2DAW2C.TELEFONS_T('958252177', '661253718')</td></tr></tbody></table>	NOM	TELEFONS	David Peñalver	UF4M2DAW2C.TELEFONS_T('958252177', '661253718')	<table><thead><tr><th>NOM</th><th>TELEFONS</th></tr></thead><tbody><tr><td>David Peñalver</td><td>UF4M2DAW2C.TELEFONS_T('958252176', '661253718')</td></tr></tbody></table>	NOM	TELEFONS	David Peñalver	UF4M2DAW2C.TELEFONS_T('958252176', '661253718')
NOM	TELEFONS								
David Peñalver	UF4M2DAW2C.TELEFONS_T('958252177', '661253718')								
NOM	TELEFONS								
David Peñalver	UF4M2DAW2C.TELEFONS_T('958252176', '661253718')								

Ex 14. Eliminar un telefon de David Peñalver (amb PL/SQL).

```
create or replace procedure eliminartelefon(nif_empl in varchar2, telefon in varchar2) as
taula_af telefons_T;
i number(5);
fi boolean;
temp number(5);

begin
  select telefons into taula_af from Empleats where nif=nif_empl;
  i:=1;
  fi:=false;
  while (i<=taula_af.COUNT and not fi) loop
    if (taula_af(i) = telefon) then
      temp:=taula_af(i);
      taula_af(i):=taula_af(taula_af.COUNT);
      taula_af(taula_af.COUNT):=temp;
      taula_af.TRIM;
      fi:=true;
    end if;
    i:=i+1;
  end loop;
  update Empleats set telefons=taula_af where nif=nif_empl;
end eliminartelefon;

execute eliminartelefon('21188594L','661253718');
```

El mètode DELETE no és pot aplicar al tipus VARRAY. Per esborrar el que faig és posar l'element a esborrar a l'última posició i després esborrar-lo amb el mètode TRIM.

Abans de l'operació		Després de l'operació	
NOM	TELEFONS	NOM	TELEFONS
David Peñalver	UF4M2DAW2C.TELEFONS_T('958252176','661253718')	David Peñalver	UF4M2DAW2C.TELEFONS_T('958252176')

Ex 15. Mostrar les aficions de tots els empleats (amb PL/SQL).

```
create or replace procedure mostrarAficions as
cursor c1 is select nif,nom,aficions from Empleats;
vreg c1%ROWTYPE;
begin
  for vreg in c1 loop
    DBMS_OUTPUT.PUT_LINE('Aficions de : ' || vreg.nom || '(' || vreg.nif || ')');
    for aficio in vreg.aficions.FIRST .. vreg.aficions.LAST loop
      DBMS_OUTPUT.PUT_LINE(' ' || vreg.aficions(aficio));
    end loop;
    DBMS_OUTPUT.PUT_LINE('');
  end loop;
end mostrarAficions;
```

NIF	NOM	AFICIONS
39671221X	Josep Maria Moncusí	UF4M2DAW2C.HOBIES_T('Viatjar')
39688127R	Borja Vallejo	UF4M2DAW2C.HOBIES_T('Submarinisme','Tenis taula','Natació','Boxa','Futbol')
39672153B	Victoria Montalvo	UF4M2DAW2C.HOBIES_T('Jocs de rol','Paddle')
21188594L	David Peñalver	UF4M2DAW2C.HOBIES_T('Paddle Surf','Escacs','Cinema','Teatre')

Aficions de : Josep Maria Moncusí (39671221X)
Viatjar

Aficions de : Borja Vallejo (39688127R)
Submarinisme
Tenis taula
Natació
Boxa
Futbol

Aficions de : Victoria Montalvo (39672153B)
Jocs de rol
Paddle

Aficions de : David Peñalver (21188594L)
Paddle Surf
Escacs
Cinema
Teatre

Implementació de les relacions entre classes

Atributs multivaluats

Una classe pot tenir un atribut que sigui compost. Aquest atribut compost no és més que un element d'una altra classe. Per exemple podem definir una classe adreça que tingui com a atributs el carrer, número, escala, pis, porta, codi postal, poble, ciutat. Després podem afegir a la nostra classe empleat (Empleat_T) un atribut anomenat domicili que sigui de tipus adreça.

```
alter type Empleat_T add attribute (domicili adreça) cascade;
```

Relacions d'associació

Les relacions d'associació són les més habituals. Aquest tipus de relacions estan definides per un 'verb'. Per exemple podríem tenir una classe àrees (Area_T) i una altra d'hospitals (Hospital_T). Un hospital pertany a un àrea (només una) i en una àrea tenim molts hospitals. El verb que em defineix la relació és 'pertany'.

En aquest tipus de relacions les 2 entitats són independents; es a dir si s'elimina un àrea els hospitals associats no tenen perquè desapareixer (es poden assignar a un altre àrea existent o nova).

La relació té cardinalitat 1:N (en un àrea hi ha N hospitals però un hospital pertany a 1 àrea). La implementació d'aquest tipus de relacions es fan definint una referència en la part de la N (en els hospitals en el nostre cas). Així definim un atribut en el tipus Hospital_T que sigui una referència al tipus Area_T (hospital REF Area_T). Amb aquest enllaç podem obtenir fàcilment l'àrea d'un hospital mitjançant la referència. Direm que en aquest cas tenim navegació des de la classe Hospital_T cap a la classe Area_T.

En algunes ocasions ens interessa la navegació des de la part de l'1 fins a l'N. En el nostre cas això vol dir que ens interessa conèixer tots els hospitals d'un àrea concreta. Tenim diverses alternatives per implementar aquesta opció. Si supossem que cap àrea pot tenir més d'un màxim d'hospitals (cosa molt raonable) aleshores definiríem un atribut hospitals en la classe Area_T que sigui un VARRAY d'objectes de la classe Hospital_T. Implementació de l'exemple:

```
create type hospital_VA as VARRAY(10) of REF Hospital_T;
create type Area_T ..... hospitals hospital_VA, ..... ;
```

També podem suposar que un àrea pot tenir un nombre arbitrari d'hospitals. En aquest cas utilitzarem una taula anidada per tenir els hospitals d'un àrea. La implementació la faríem així:

```
create type hospital_NT as TABLE of REF Hospital_T;
create type Area_T ..... hospitals hospital_NT, ..... ;
```

Si ens fixem veiem que independentment del tipus de col·lecció utilitzat el que s'emmagatzemen són referències a objectes. Això és així perquè les classes hospital i àrea són independents entre sí i es poden relacionar amb altres classes.

Aquest exemple d'associació és una relació 1:N. Les relacions d'associació molts a molts (M:N) es descomposem creant una classe que representa a la relació (a més aquí aniríem atributs que són simultàniament de les dues classes relacionades) i relacionant cada classe amb la nova classe creada. Les relacions entre cada classe i la nova creada són 1:N.

Per exemple podem tenir una classe metges que es relaciona amb els hospitals. Podem suposar que un metge pot treballar en més d'un hospital i evidentment en un hospital treballen molts metges. En aquest cas crearíem una classe Treballa_T amb dos referències una cap un objecte de la classe Hospital_T i l'altra cap a la classe Area_T. Ara la relació entre Hospital_T i Treballa_T és 1:N i la relació entre Area_T i Treballa_T és 1:N també. Opcionalment es pot tenir camps de tipus col·lecció a les classes Hospital_T i Area_T.

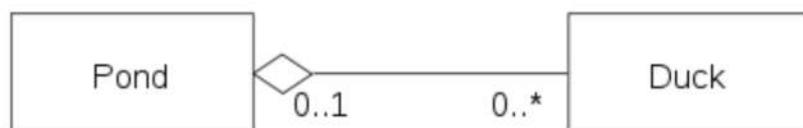
Tindrem una taula d'objectes per a cada classe que intervé en la relació. En les taules es definiran les claus primàries i opcionalment es poden definir les foranes. Degut a la potència del llenguatge de consultes del model relacional en molts casos no cal definir atributs de tipus col·lecció en les classes. En qualsevol cas això és una qüestió que depén del model de dades que es vol tenir.

Relacions d'agregació

Entre dues classes hi ha una classe està formada per diferents parts d'elements de l'altra classe però els objectes de la classe que modelen les parts poden existir sense la classe de la qual depenen. La destrucció d'un objecte del tot no implica la destrucció de les seves parts. Per exemple entre Empreses i clients podríen tenir una relació d'agregació de manera que es pot considerar que una empresa agrupa o consta de diferents clients. Quan

s'elimina una empresa no s'eliminen els seus clients. Les relacions d'agregació sempre són 1:N amb 1 a la part del tot i N a la banda de les parts.

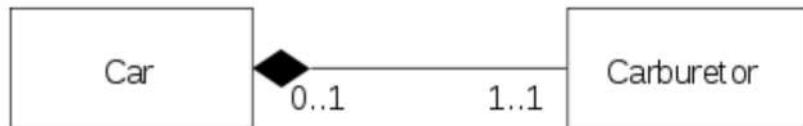
En el llenguatge UML aquest tipus de relació és representat amb un rombe en blanc a la part del tot com es mostra al següent exemple.



Relacions de composició

Les relacions de composició també relacionen parts amb un tot però és un tipus de relació més forta que la d'agregació. Si eliminem un registre de la classe que representa el tot s'esborren tots els registres relacionats. Els components del tot no tenen sentit sense el tot. Per exemple podem tenir una classe factura que està composta per línies de factura. Quan s'elimina una factura automàticament s'eliminen totes les línies de factura associades. No té sentit tenir línies de factura sense una factura associada. En aquest cas s'utilitza un atribut col·lecció (varray o nested table) per exemple llinesfactur en la classe factura amb elements de tipus línia factura (no referències a línies de factura). Aquest tipus de relació és sempre de cardinalitat 1:N i la navegació es fa sempre des del tot fins a les parts. Les línies de factura només són accessibles des de les factures i no es relacionen amb cap altra classe. Crearem una taula amb objectes de la classe factura i cada factura tindrà un atribut que serà una col·lecció de línies de factura.

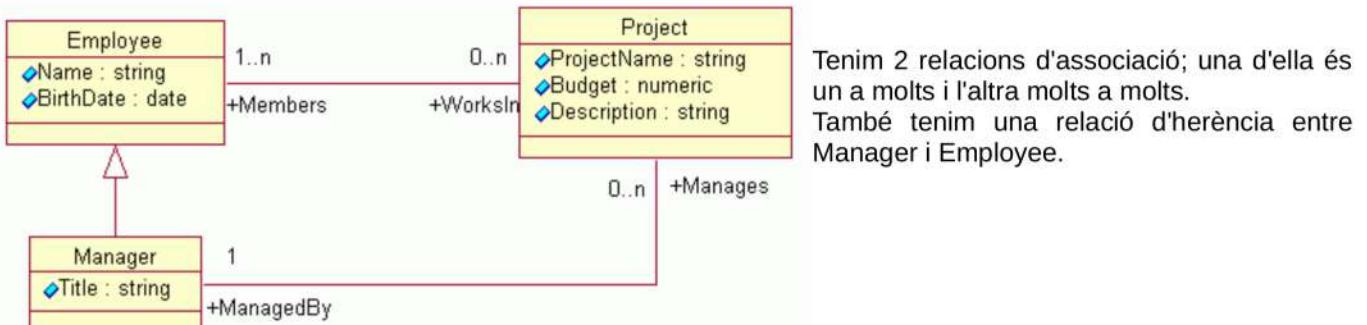
S'utilitza un rombe de color negre a la banda del tot tal com es mostra a la següent imatge.



Relacions d'herència

Entre 2 classes hi ha una relació d'herència quan els elements de la subclasse també són realment elements de la superclasse. Per exemple si tenim dues classes una d'estudiants i una altra de professors aquestes 2 es poden considerar subclasses de la superclasse persones. Quan un element només pot pertànyer a una de les subclasses aleshores es pot optar per tenir una única taula amb objecte d'elements de la superclasses amb la qual es podran inserir tant elements de la superclasse com de les subclasses. Per exemple si una persona o és professor o és alumne però no les dues coses simultàniament aleshores tindriem una única taula de personnes però si això no és així necessitarem una taula per a la superclasse i les subclasses i en cada subclass tindrem un atribut que serà una referència a la superclasse.

A continuació es mostra un diagrama UML on apareixen relacions d'associació (bidireccionals) i herència.



Quan s'implementa una relació bidireccional entre classes tenim que en la definició de la primera classe necessitem que la segona classe existeixi i en la definició de la segona necessitem que la primera existeixi. Així entrem en un bucle que sembla no tingui solució. Afortunadament aquest problema es resolt utilitzant el mecanisme de definició incompleta de les classes. Per mostrar com funciona aquest mecanisme tornem a l'exemple amb el qual hem estat treballant. Suposem que definim a la classe Departament_T un atribut anomenat empleats que és una col·lecció (una nested table per exemple) de referències (relació d'associació) a objectes Empleat_T. Ara per definir aquesta classe necessitem que estigui creada la classe Empleat_T. Però, per definir la classe Empleat_T ha d'existir la classe Departament_T ja que l'atribut departament és una referència a un objecte

d'aquesta última classe. Amb la definició incompleta de classes es faria així.

1	<code>CREATE TYPE Departament_T; /* Definició incompleta de la classe Departament_T */</code>
2	<code>CREATE TYPE Empleat_T as OBJECT (nif varchar2(9), nom varchar2(80), departament REF Departament_T);</code>
3	<code>CREATE TYPE Empleat_NT as TABLE of REF Empleat_T;</code>
4	<code>CREATE TYPE Departament_T as OBJECT (codi varchar2(4), nom varchar2(80), empleats Empleat_NT);</code>
5	<code>CREATE TABLE Departaments of Departament_T (primary key (codi)) nested table empleats store as taula_empleats;</code>
6	<code>CREATE TABLE Empleats of Empleat_T (primary key (nif), foreign key (departament) references Departaments);</code>

Definició mètodes

Herència i sobreescritura de mètodes

Ara veurem com funciona la sobreescritura de mètodes. Anteriorment van veure com es definia una classe per a que permeti ser heredada per altres classes. Ara crearem en Oracle les classes definides en Java que es presenten a continuació i se sobreescriviran els mètodes que apareixen sobreescrits en aquestes classes.

```
public class Vehiculo {  
    private int matricula;  
    private int potencia;  
  
    public Vehiculo (int matricula, int potencia) {  
        this.matricula = matricula;  
        this.potencia = potencia;  
    }  
    public int getMatricula() { return matricula; }  
    public int getPotencia() { return potencia; }  
    public void displayDatos() {  
        System.out.println("Matricula: " + matricula);  
        System.out.println("Potencia: " + potencia);  
    }  
}  
  
public class Coche extends Vehiculo {  
    private int numeroPlazas;  
  
    public Coche(int matricula, int potencia, int numeroPlazas) {  
        super(matricula, potencia);  
        this.numeroPlazas = numeroPlazas;  
    }  
    public int getNumeroPlazas() { return numeroPlazas; }  
    public void displayDatos() {  
        super.displayDatos();  
        System.out.println("Plazas: " + numeroPlazas);  
    }  
}
```

```
public class Cabina {  
    private int potencia;  
  
    public Cabina(int potencia) {  
        this.potencia = potencia;  
    }  
    public int getPotencia() { return potencia; }  
}  
  
public class Remolque {  
    private int cargaMaxima;  
  
    public Remolque(int cargaMaxima) {  
        this.cargaMaxima = cargaMaxima;  
    }  
    public int getCargaMaxima() { return cargaMaxima; }  
}  
  
public class Camion extends Vehiculo {  
    private Cabina cabina;  
    private Remolque remolque;  
  
    public Camion(int matricula, Cabina cabina, Remolque remolque) {  
        super(matricula, cabina.getPotencia());  
        this.cabina = cabina;  
        this.remolque = remolque;  
    }  
    public int getCargaMaxima() { return remolque.getCargaMaxima(); }  
    public void displayDatos() {  
        super.displayDatos();  
        System.out.println("Carga máxima: " + remolque.getCargaMaxima());  
    }  
}
```

```
}
```

A més definirem una taula de Vehiculos anomenada **Vehiculos_tab** on desarem qualsevol tipus de vehicle. A més implementarem mètodes statics per inserir, modificar i eliminar qualsevol tipus de vehicle en la taula.

Un mètode static en Oracle es defineix de manera totalment anàloga a com es fa en Java. Si és un procediment encapsularem aquest amb **static procedure** i si és una funció amb **static function**.

La crida a un procediment o funció estàtica es fa com en Java; es posa el nom de la classe seguit d'un punt i el nom del mètode. Per exemple per cridar al mètode insertar faríem: **Camion.insertar(c Camion)**

Especificació de les classes

```
create or replace type Vehiculo as object
(
    matricula varchar2(9),
    potencia number(6),
    member function getMatricula return varchar2,
    member function getPotencia return number,
    member procedure displayDatos,
    static procedure insertar(v Vehiculo),
    static procedure modificar(v Vehiculo),
    static procedure eliminar(mat varchar2)
) not final;

create or replace type Coche under Vehiculo
(
    numeroPlazas number(2),
    member function getNumeroPlazas return number,
    overriding member procedure displayDatos,
    static procedure insertar(v Coche),
    static procedure modificar(v Coche)
);
```

<pre> create or replace type Cabina as object (potencia number(6), member function getPotencia return number); </pre>	<pre> create or replace type Remolque as object (cargaMaxima number(6), member function getCargaMaxima return number); </pre>
<pre> create or replace type Camion under Vehiculo (cbna Cabina, rmq Remolque, constructor function Camion(mat varchar2,cbna Cabina,rmq Remolque) return SELF as RESULT, member function getCargaMaxima return number, overriding member procedure displayDatos, static procedure insertar(v Camion), static procedure modificar(v Camion)); </pre>	

Creació de la taula de vehicles

```

create table Vehiculos_tab of Vehiculo
(
    primary key (matricula)
);

```

Implementació de les classes

```

create or replace type body Vehiculo as

member function getMatricula return varchar2 as
begin
    return SELF.matricula;
end getMatricula;

member function getPotencia return number as
begin
    return SELF.potencia;
end getPotencia;

member procedure displayDatos as
begin
    SYS.DBMS_OUTPUT.PUT_LINE('Matrícula: ' || SELF.matricula);
    SYS.DBMS_OUTPUT.PUT_LINE('Potència: ' || SELF.potencia);
end displayDatos;

static procedure insertar(v Vehiculo) as
begin
    insert into Vehiculos_tab values(v);
end insertar;

static procedure modificar(v Vehiculo) as
begin
    update Vehiculos_tab E set E = v where E.matricula = v.matricula;
end modificar;

static procedure eliminar(mat varchar2) as
begin
    delete from Vehiculos_tab where matricula=mat;
end eliminar;

end;

```

```

create or replace type body Coche as

member function getNumeroPlazas return number as
begin
    return SELF.numeroPlazas;
end getNumeroPlazas;

overriding member procedure displayDatos as
begin
    (SELF as Vehiculo).displayDatos();
    SYS.DBMS_OUTPUT.PUT_LINE('Número de places: ' || SELF.numeroPlazas);
end displayDatos;

static procedure insertar(v Coche) as
begin
    insert into Vehiculos_tab values(v);
end insertar;

static procedure modificar(v Coche) as
begin
    update Vehiculos_tab E set E = v where E.matricula = v.matricula;
end modificar;

end;

create or replace type body Cabina as

member function getPotencia return number as
begin
    return SELF.potencia;
end getPotencia;

end;

create or replace type body Remolque as

member function getCargaMaxima return number as
begin
    return SELF.cargaMaxima;
end getCargaMaxima;

end;

```

```

create or replace type body Camion as

constructor function Camion(mat varchar2,cbna Cabina,rmq Remolque) return SELF as RESULT as
begin
    SELF.matricula:=mat;
    SELF.potencia:=cbna.getPotencia();
    SELF.cbna:=cbna;
    SELF.rmq:=rmq;
    return;
end Camion;

member function getCargaMaxima return number as
begin
    return rmq.getCargaMaxima();
end getCargaMaxima;

overriding member procedure displayDatos as
begin
    (SELF as Vehiculo).displayDatos();
    SYS.DBMS_OUTPUT.PUT_LINE('Càrrega màxima: ' || rmq.getCargaMaxima());
end displayDatos;

static procedure insertar(v Camion) as
begin
    insert into Vehiculos_tab values(v);
end insertar;

static procedure modificar(v Camion) as
begin
    update Vehiculos_tab E set E = v where E.matricula = v.matricula;
end modificar;

end;

```

Exemples operacions amb la classe Vehiculo

Aquest exemple mostra l'accés als mètodes propis de la classe Coche i la seva superclasse Vehiculo. A més es crida al mètode estàtic insertar que insereix el Coche a la classe de Vehiculos.

```

declare
v1 Coche;
begin
    v1:=new Coche('21345-AFZ',1600,4);
    v1.displayDatos();
    DBMS_OUTPUT.PUT_LINE(");
    DBMS_OUTPUT.PUT_LINE('Matrícula = ' || v1.getMatricula());
    DBMS_OUTPUT.PUT_LINE('Potència = ' || v1.getPotencia());
    DBMS_OUTPUT.PUT_LINE('Num. places = ' || v1.getNumEROPlazas());
    Coche.insertar(v1);
end;

```

Salida de DBMS	
	Tamaño de Buffer: 20000
uf4m2daw2d x	
Matrícula:	21345-AFZ
Potència:	1600
Número de places:	4
Matrícula =	21345-AFZ
Potència =	1600
Num. places =	4
MATRICULA	POTENCIA
21345-AFZ	1600

Els mètodes estàtics modificar de la classe Coche i eliminar de la seva superclasse Vehiculo també funcionen correctament quan treballem amb objectes de la classe Coche.

```

declare
v1 Coche;
begin
    v1:= new Coche('21345-AFZ',1660,5);
    Coche.modificar(v1);
end;
begin
    Coche.eliminar('21345-AFZ');
end;

```

MATRICULA	POTENCIA
21345-AFZ	1660
MATRICU...	POTENCIA

Quan assignem un objecte de la classe Coche a una variable de la classe Vehiculo només seran accessibles els mètodes

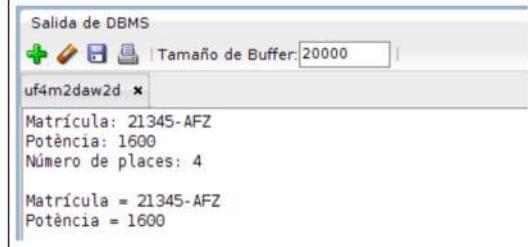
sobreescrits (displayDatos()) en la classe Coche i els mètodes de la classe Vehiculo.

```
declare
v1 Vehiculo;
begin
    v1:=new Coche('21345-AFZ',1600,4);
    v1.displayDatos();
    DBMS_OUTPUT.PUT_LINE('');
    DBMS_OUTPUT.PUT_LINE('Matrícula = ' || v1.getMatricula());
    DBMS_OUTPUT.PUT_LINE('Potència = ' || v1.getPotencia());
    DBMS_OUTPUT.PUT_LINE('Num. places = ' || v1.getNumPlazas());
end;
```

```
Informe de error -
ORA-06550: linea 9, columna 45:
PLS-00302: el componente 'GETNUMEROPLAZAS' se debe declarar
ORA-06550: linea 9, columna 1:
PL/SQL: Statement ignored
06550. 00000 -  "line %s, column %s:\n%s"
*Cause:  Usually a PL/SQL compilation error.
*Action:
```

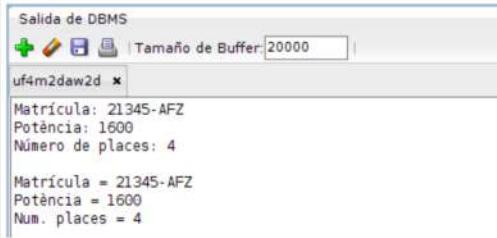
L'exemple que tenim damunt falla perquè es crida al mètode getNumeroPlazas() que no està sobreescrit i pertany a la classe Coche. El següent exemple mostra com el mateix codi sí funciona quan treiem el mètode getNumeroPlazas().

```
declare
v1 Vehiculo;
begin
    v1:=new Coche('21345-AFZ',1600,4);
    v1.displayDatos();
    DBMS_OUTPUT.PUT_LINE('');
    DBMS_OUTPUT.PUT_LINE('Matrícula = ' || v1.getMatricula());
    DBMS_OUTPUT.PUT_LINE('Potència = ' || v1.getPotencia());
end;
```



Mitjançant l'operador treat podem accedir als mètodes propis (no sobreescrits) definits a la classe Coche.

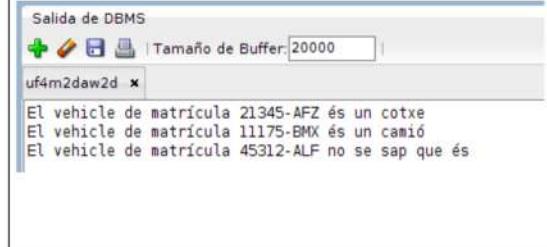
```
declare
v1 Vehiculo;
begin
    v1:=new Coche('21345-AFZ',1600,4);
    v1.displayDatos();
    DBMS_OUTPUT.PUT_LINE('');
    DBMS_OUTPUT.PUT_LINE('Matrícula = ' || v1.getMatricula());
    DBMS_OUTPUT.PUT_LINE('Potència = ' || v1.getPotencia());
    DBMS_OUTPUT.PUT_LINE('Num. places = ' || treat(v1 as Coche).getNumPlazas());
end;
```



El següent bloc de codi PL/SQL assigna a una variable de la classe Vehiculo els diferents vehicles que hi ha (Coche i Camion) i mostra la informació de cada vehicle fent ús de l'operador is of.

```
create or replace procedure mostrarVehiculo(v Vehiculo) as
begin
    if (v is of (Coche)) then
        DBMS_OUTPUT.PUT_LINE('El vehicle de matrícula ' || v.getMatricula() || ' és un cotxe');
    elsif ((v is of (Camion))) then
        DBMS_OUTPUT.PUT_LINE('El vehicle de matrícula ' || v.getMatricula() || ' és un camió');
    else
        DBMS_OUTPUT.PUT_LINE('El vehicle de matrícula ' || v.getMatricula() || ' no se sap que és');
    end if;
end mostrarVehiculo;
```

```
declare
v1 Vehiculo;
begin
    v1:=new Coche('21345-AFZ',1600,4);
    mostrarVehiculo(v1);
    v1:=new Camion('11175-BMX',Cabina(5000),Remolque(30000));
    mostrarVehiculo(v1);
    v1:=new Vehiculo('45312-ALF',2100);
    mostrarVehiculo(v1);
end;
```



Salida de DBMS
uf4m2daw2d x | Tamaño de Buffer 20000
El vehicle de matrícula 21345-AFZ és un cotxe
El vehicle de matrícula 11175-BMX és un camió
El vehicle de matrícula 45312-ALF no se sap que és