DRIVER V1.34
Manuel de Référence

Franck LEBASTARD
CERMICS
INRIA
06902 Sophia-Antipolis Cedex
France

Résumé

Ce document constitue la documentation du système DRIVER[1] v1.34, couche objet virtuelle persistante permettant d'utiliser dans un même formalisme objet choisi aussi bien l'information contenue dans les bases de données relationnelles que la connaissance d'un système de plus haut niveau, comme notre générateur de Systèmes Experts SMECI. Un schéma de correspondances, défini par l'utilisateur, associe aux données des bases connectées une représentation objet qui permet de les manipuler et de les exploiter de façon absolument transparente dans l'environnement du système expert, notamment dans le cadre du raisonnement. DRIVER permet également d'apporter la persistance aux objets de l'environnement, au gré de l'utilisateur.

DRIVER V1.34
Reference Manual

Abstract

This document is the reference manual of DRIVER v1.34, a persistent virtual object layer, that permits to use, in a same chosen object formalism, both the information contained in relational databases and the knowledge of a higher-level system, such as our expert system shell SMECI. A user-defined mapping assigns an object representation to data of connected bases; it permits to handle and to utilize them exactly as other objects in the expert system environment, for example during reasoning. DRIVER can also supply some environment objects with persistency, according to user's wishes.

1. **D**onnées **R**elationnelles et **I**nterface **V**irtuelle pour l'**E**xpertise et le **R**aisonnement

Ce document présente l'interface fonctionnelle de DRIVER version 1.34. Une présentation générale du système peut être trouvée dans [Leb90a, Leb90b, Leb92], une description très détaillée (fonctionnalités, structures et algorithmes) dans [Leb93].

- Sont présentées dans une première partie toutes les fonctions de construction du schéma de correspondances, de manipulation et de suppression de ses composants. Les trois macros DRIVER-DEFTABLE, DRIVER-DEFCLASS et DRIVER-DEF CLASSMAP sont essentielles pour l'utilisateur débutant. Elles permettent à elles seules la description d'un schéma complet (voir l'exemple de schéma §6).

- Sont ensuite présentées toutes les primitives de gestion de la couche objet virtuelle, la connexion aux SGBD, le filtrage d'objets dans la base, le chargement et l'écriture des objets.

- La prise en compte d'un nouveau modèle objet est décrite en §4, une illustration complète pour SMECI en §8.

- Un exemple de base de données, de schéma de correspondances et de session utilisateur utilisant cette base et ce schéma suivent.

- L'index des erreurs DRIVER est donné en §9.

- Enfin, l'index des fonctions et variables est donné en §10.

Références

[Leb90a] F. Lebastard. DRIVER : A persistent virtual object layer for reasoning. In *ISMIS-90, 5th International Symposium on Methodologies for Intelligent Systems*, pages 74–81, Knoxville (Tennessee), October 1990.

[Leb90b] F. Lebastard. DRIVER : A persistent virtual object layer for reasoning. In *AAAI Workshop on Knowledge Base Management Systems*, page multiple, Boston (Massachusetts), July 1990.

[Leb92] F. Lebastard. DRIVER : Une couche objet pour les bases de données relationnelles. Technical Report 92-6, CERMICS-INRIA, Sophia-Antipolis (France), Octobre 1992. 19 pages.

[Leb93] F. Lebastard. *DRIVER : Une couche objet virtuelle persistante pour le raisonnement sur les bases de données relationnelles.* Thèse de doctorat, INSA de Lyon/INRIA/CERMICS Sophia-Antipolis, Mars 1993. 380 pages.

DRIVER **[FEATURE]**

Ce trait indique que le module DRIVER est chargé en mémoire. Si ce n'est pas le cas, on le charge par `^Ldriver`. Le fichier `driver.ll` doit se trouver dans le répertoire courant.

Exemple :

```
? (FEATUREP 'driver)
= ()
? ^Ldriver
Chargement d'ASQUELL...
Chargement d'ASQUELL oK
Chargement de DRIVER...
DRIVER v1.34
Chargement de DRIVER oK
= driver.ll
? (FEATUREP 'driver)
= t
```

1 Construction d'un schéma de correspondances

Les fonctions suivantes permettent de construire un schéma de correspondances à partir de la description des éléments connus des représentations relationnelle et objet à associer. Si les deux représentations et leur couplage sont entièrement décrites, le schéma est complet et directement exploitable. Si leur description n'est que partielle, des primitives permettent de générer les éléments manquants et de compléter le schéma.

1.1 Création et recherche d'un schéma de correspondances

(DRIVER-CREATE-MAPPING <mapping-name> . <overwritep>)
[SUBR à 1 ou 2 arguments]

Crée et retourne en valeur un nouveau schéma de correspondances de nom <mapping-name>. Ce schéma doit être complété au moyen des fonctions du paragraphe suivant avant toute utilisation. Si le schéma est le premier créé lors de la session, il devient automatiquement le schéma courant.

Si la variable système DRIVER-AFFECT-CURSOR-P vaut t (valeur par défaut) et si au moins une connexion vers une base a été établie, un curseur est affecté au

nouveau schéma. En cas de redéfinition d'un schéma existant, le curseur courant de ce schéma est préservé.

En cas de tentative de redéfinition d'un schéma, l'erreur "schéma déjà existant" est provoquée, sauf si <overwrite> est précisé à t. Dans ce cas, l'ancienne définition est perdue, écrasée par la nouvelle, ainsi que toutes les définitions qui y étaient rattachées (sauf son éventuel curseur). Toutes les structures qui la contenaient pointent maintenant sur la nouvelle.

Exemple:

```
? (DRIVER-CREATE-MAPPING 'map1)
= map1
? (DRIVER-CREATE-MAPPING 'map1)
** driver-create-mapping : existent mapping : map1
? (DRIVER-CREATE-MAPPING 'map1 t)
= map1
```

DRIVER-AFFECT-CURSOR-P **[Variable]**

Si une connexion à une base de données a été préalablement établie au moyen de la fonction DRIVER-CONNECT, la valeur de DRIVER-AFFECT-CURSOR-P conditionne l'affectation d'un curseur courant aux schémas de correspondances au moment de leurs créations. Si elle vaut t (valeur par défaut), l'affectation est faite. Le curseur choisi est le curseur courant du schéma courant s'il existe, ou celui retourné par la première exécution de DRIVER-CONNECT dans le cas contraire.

(DRIVER-FIND-MAPPING <mapping-name>) **[SUBR à 1 argument]**

Retourne le schéma de correspondances de nom <mapping-name> s'il existe.

Exemple:

```
? (DRIVER-FIND-MAPPING 'map1)
= map1
? (DRIVER-FIND-MAPPING 'map2)
= ()
```

(DRIVER-MAPPING-P <mapping>) **[SUBR à 1 argument]**

Ce prédicat est vrai et renvoie l'objet <mapping> si ce dernier est un schéma de correspondances.

Exemple:

```
? (DRIVER-MAPPING-P 'map1)
= ()
? (DRIVER-MAPPING-P (DRIVER-FIND-MAPPING 'map1))
= map1
```

(DRIVER-CURRENT-MAPPING <mapping-name>)
(DRIVER-CURRENT-MAPPING0 <mapping>) [SUBR à 0 ou 1 argument]

Variables-fonctions permettant d'accéder au schéma de correspondances courant si <mapping-name> (resp. l'objet <mapping>) n'est pas fourni, et de le définir dans le cas contraire.

Exemple:

```
? (DRIVER-CURRENT-MAPPING)
= map1       ; initialise par le premier DRIVER-CREATE-MAPPING
? (DRIVER-CREATE-MAPPING 'map2)
= map2
? (DRIVER-CURRENT-MAPPING)
= map1
? (DRIVER-CURRENT-MAPPING 'map2)
= map2
? (DRIVER-CURRENT-MAPPING)
= map2
? (DRIVER-CURRENT-MAPPING0 (DRIVER-FIND-MAPPING 'map1))
= map1
? (DRIVER-CURRENT-MAPPING0)
= map1
```

(DRIVER-ALL-MAPPINGS <all-p>) [SUBR à 0 ou 1 argument]

Sans argument, cette fonction renvoie la liste de tous les schémas de correspondances de l'utilisateur créés depuis le début de la session. Si le paramètre t est précisé, tous les schémas seront renvoyés, y compris les schémas système (le méta-schéma, par exemple).

Exemple:

```
? (DRIVER-ALL-MAPPINGS)
= (map1 map2)
? (DRIVER-ALL-MAPPINGS t)
= (map1 map2)
? (DRIVER-METAMAPPING)
= driver-metamapping
? (DRIVER-ALL-MAPPINGS t)
= (map1 map2 driver-metamapping)
```

(DRIVER-COMPILE-MAPPING <mapping-name>)
(DRIVER-COMPILE-MAPPING0 <mapping>) **[SUBR à 1 argument]**

Compile le schéma de correspondances de nom <mapping-name> (resp. l'objet <mapping>). Cette opération, qui accélère les échanges d'objets entre l'environnement et la base de données, ne doit être faite que quand le schéma est terminé. Toute nouvelle modification du schéma annihile sa compilation.

Exemple:

```
? (DRIVER-COMPILE-MAPPING 'map1)
= t
```

(DRIVER-DELETE-MAPPING <mapping-name>)
(DRIVER-DELETE-MAPPING0 <mapping>) **[SUBR à 1 argument]**

Supprime le schéma de correspondances de nom <mapping-name> (resp. l'objet <mapping>) dans l'environnement. Toutes les données qu'il contient sont également perdues.

Exemple:

```
? (DRIVER-DELETE-MAPPING 'map2)
= t
```

1.2 Description de la représentation relationnelle

Les fonctions suivantes s'appliquent toutes au schéma de correspondances courant.

Elles permettent de décrire la représentation relationnelle et de définir dans le schéma des tables et des attributs.

(DRIVER-DEFTABLE <table-name> <attr1> ... <attrN>) **[MACRO]**

définit dans le schéma courant une table de nom <table-name> (symbole), comportant les attributs <attr1> ... <attrN>. L'ordre utilisé pour décrire ces attributs sera repris pour l'éventuelle création de la table considérée. Cette macro crée une table et chacun de ses attributs spécifiés.

<attr1> ... <attrN> décrivent les attributs de la table. Ce sont des listes à 4 éléments ayant la structure suivante :

(<attribute-name> <attr-type> <type-length> <status>)

Le type de l'attribut, la longueur de ce type ainsi que le statut de l'attribut doivent être spécifiés.

Les types d'attribut possibles sont par défaut *integer*, *float* et *string*. Cette liste peut être complétée en fonction de la base de données utilisée et des types qu'elle reconnaît au moyen de la fonction DRIVER-ADD-ATTRIBUTE-TYPE. Elle est consultée à chaque création d'attribut.

Les différents statuts possibles sont :

unique Il ne peut exister deux n-uplets de la table ayant même valeur (différente de *null*) pour l'attribut en question.

not-null L'attribut ne peut valoir *null*. Un nouveau n-uplet qu'on insère dans la table doit nécessairement définir une valeur pour l'attribut en question.

keypart L'attribut fait partie de la clé de la table.

void ou () L'attribut est classique. Il n'a pas de statut particulier.

Si la variable globale DRIVER-TABLE-MAKE vaut t (la valeur par défaut est nil), la fonction DRIVER-MAKE-TABLE0 est appelée et se traduit par la définition dans le SGBD courant de la relation correspondante.

DRIVER-DEFTABLE peut être décrit en lisp de la manière suivante (les fonctions DRIVER-CREATE-TABLE et DRIVER-CREATE-ATTRIBUTE0 sont explicitées ci-après) :

```
(DEFMACRO DRIVER-DEFTABLE (name . attr-list)
    `(LET ((table (DRIVER-CREATE-TABLE ',name t)))
          ,@(MAPCAR (LAMBDA (attr)
                      `(DRIVER-CREATE-ATTRIBUTE0 table
                                                 ,@(MAPCAR 'KWOTE attr)))
                    attr-list)
          (IF DRIVER-TABLE-MAKE
              (DRIVER-MAKE-TABLE0 table))
          table))
```

Exemple:

```
? (DRIVER-DEFTABLE emp
?                  (empno integer  2 keypart)
?                  (ename string  20 not-null)
?                  (fname string  18 ())
?                  (job   string  20 ())
?                  (sal   float    8 ())
?                  (mgr   integer  2 ())
?                  (dpt   integer  2 ()))
= emp
```

```
? (DRIVER-DEFTABLE vehicle
?                  (vnum    integer  2 keypart)
?                  (model   string  15 ())
?                  (licence string  15 unique))
= vehicle
```

DRIVER-TABLE-MAKE **[Variable]**

Si la variable globale DRIVER-TABLE-MAKE vaut t (valeur par défaut), la fonction DRIVER-MAKE-TABLE0 est exécutée à la fin de la déclaration d'une table par DRIVER-DEFTABLE et se traduit par la définition dans le SGBD courant de la relation correspondante.

(DRIVER-CREATE-TABLE <table-name> . <overwritep>)
[SUBR à 1 ou 2 arguments]

Définit la table de nom <table-name> dans le schéma courant. Cette définition est initialement vide d'attribut. Les attributs correspondants devront être créés au moyen de la fonction DRIVER-CREATE-ATTRIBUTE.

En cas de tentative de redéfinition d'une table, l'erreur "table déjà existante" est provoquée, sauf si <overwrite> est précisé à t. Dans ce cas, l'ancienne définition est perdue, écrasée par la nouvelle, ainsi que toutes les définitions d'attributs qui l'utilisaient. Toutes les structures qui la contenaient pointent maintenant sur la nouvelle.

Exemple:

```
? (DRIVER-CREATE-TABLE 'person)
= person
? (DRIVER-CREATE-TABLE 'address)
= address
? (DRIVER-CREATE-TABLE 'dept)
= dept
? (DRIVER-CREATE-TABLE 'company)
= company
```

(DRIVER-FIND-TABLE <table-name>) **[SUBR à 1 argument]**

Retourne la table de nom <table-name> si elle existe dans le schéma courant.

Exemple:

```
? (DRIVER-FIND-TABLE 'person)
= person
```

(DRIVER-TABLE-P <table>) **[SUBR à 1 argument]**

Ce prédicat est vrai et renvoie l'objet <table> si ce dernier est une table.

Cette fonction est strictement identique à la fonction DRIVER-TABLE-VAR-P.

Exemple:

```
? (DRIVER-TABLE-P (DRIVER-FIND-TABLE 'person))
= person
```

(DRIVER-DELETE-TABLE <table-name>)
(DRIVER-DELETE-TABLE0 <table>) **[SUBR à 1 argument]**

Détruit la table de nom <table-name> (resp. l'objet <table>) dans le schéma courant (définition et variables). Par effet de bord, les attributs définis dans cette table sont également détruits. De même, toutes les références à la table sont retirées du schéma de correspondances.

Cette fonction est strictement identique à la fonction DRIVER-DELETE-TABLE-DEF.

Exemple:

```
? (DRIVER-DELETE-TABLE 'company)
= t
```

(DRIVER-CREATE-ATTRIBUTE <table-name> <attr-name>
<attr-type> <typ-len> <status> . <overwritep>)

(DRIVER-CREATE-ATTRIBUTE0 <table> <attr-name>
<attr-type> <typ-len> <status> . <overwritep>)
[SUBR à 5 ou 6 arguments]

Définit l'attribut de nom <attr-name> dans la table de nom <table-name> (resp. à l'objet <table>). Le type de l'attribut, la longueur de ce type ainsi que le statut de l'attribut doivent être spécifiés.

Les types d'attribut possibles sont par défaut *integer*, *float* et *string*. Cette liste peut être complétée en fonction de la base de données utilisée et des types qu'elle reconnaît au moyen de la fonction DRIVER-ADD-ATTRIBUTE-TYPE. Elle est consultée à chaque création d'attribut.

Les différents statuts possibles sont :

unique Il ne peut exister deux n-uplets de la table ayant même valeur (différente de *null*) pour l'attribut en question.

not-null L'attribut ne peut valoir *null*. Un nouveau n-uplet qu'on insère dans la table doit nécessairement définir une valeur pour l'attribut en question.

keypart L'attribut fait partie de la clé de la table.

void ou () L'attribut est classique. Il n'a pas de statut particulier.

En cas de tentative de redéfinition d'un attribut, l'erreur "Attribut déjà existant" est provoquée, sauf si <overwrite> est précisé à t. Dans ce cas, l'ancienne définition est perdue, écrasée par la nouvelle. Toutes les structures qui le contenaient pointent maintenant sur la nouvelle.

Exemple:

```
? (DRIVER-CREATE-ATTRIBUTE 'person 'pname 'string 20 'keypart)
= person.pname
? (DRIVER-CREATE-ATTRIBUTE 'person 'fname 'string 18 'keypart)
= person.fname
? (DRIVER-CREATE-ATTRIBUTEO
?       (DRIVER-FIND-TABLE 'person) 'ssnum 'string 15 'unique)
= person.ssnum
? (DRIVER-CREATE-ATTRIBUTE 'person 'position 'string 10 ())
= person.weight
? (DRIVER-CREATE-ATTRIBUTE 'person 'car 'integer 2 ())
= person.car
? (DRIVER-CREATE-ATTRIBUTE 'address 'empno 'integer 2 'keypart)
= address.empno
? (DRIVER-CREATE-ATTRIBUTE 'address 'state 'string 5 ())
= address.empno
? (DRIVER-CREATE-ATTRIBUTE 'dept 'deptnum 'integer 2 'keypart)
= dept.deptnum
```

(DRIVER-ATTRIBUTE-TYPES) [SUBR sans argument]

Retourne la liste des types d'attribut reconnus par DRIVER.

```
? (DRIVER-ATTRIBUTE-TYPES)
= (integer float string)
```

(DRIVER-ADD-ATTRIBUTE-TYPE <newtype>) [SUBR à 1 argument]

Définit un nouveau type d'attribut. Cette fonction permet de compléter la liste des types utilisables disponibles dans le SGBD utilisé.

ATTENTION: cette fonction ne vérifie pas que le SGBD reconnaît effectivement le type rajouté.

Exemple:

```
? (DRIVER-CREATE-ATTRIBUTE 'vehicle 'vdate 'date () ())
*** driver-create-attribute : Bad attribute type : date
? (DRIVER-ADD-ATTRIBUTE-TYPE 'date)
= date
? (DRIVER-CREATE-ATTRIBUTE 'vehicle 'vdate 'date () ())
= vehicle.vdate
```

(DRIVER-FIND-ATTRIBUTE <table-name> <attr-name>)
(DRIVER-FIND-ATTRIBUTE0 <table> <attr-name>)

[SUBR à 2 arguments]

Retourne l'attribut de nom <attr-name> de la table de nom <table-name> (de la table <table>) si elle existe.

La fonction DRIVER-FIND-ATTRIBUTE est strictement identique à la fonction DRIVER-FIND-ATTRIBUTE-VAR.

Exemple:

```
? (DRIVER-FIND-ATTRIBUTE 'emp 'empno)
= emp.empno
? (DRIVER-FIND-ATTRIBUTE 'address 'empno)
= address.empno
```

(DRIVER-ATTRIBUTE-P <attr>) **[SUBR à 1 argument]**

Ce prédicat est vrai et renvoie l'objet <attr> si ce dernier est un attribut.

Cette fonction est strictement identique à la fonction DRIVER-ATTRIBUTE-VAR-P.

Exemple:

```
? (DRIVER-ATTRIBUTE-P (DRIVER-FIND-ATTRIBUTE 'emp 'empno))
= emp.empno
```

(DRIVER-DELETE-ATTRIBUTE <table-name> <attr-name>)

[SUBR à 2 arguments]

(DRIVER-DELETE-ATTRIBUTE0 <attr>) **[SUBR à 1 argument]**

Détruit l'attribut de nom <attr-name> dans la table de nom <table-name> (resp. l'objet <attr>) dans le schéma courant (définition et variables). Toutes les références à l'attribut sont retirées du schéma de correspondances.

La fonction DRIVER-DELETE-ATTRIBUTE est strictement identique à la fonction DRIVER-DELETE-ATTRIBUTE-DEF.

Exemple:

```
? (DRIVER-DELETE-ATTRIBUTE 'vehicle 'vdate)
= t
```

(DRIVER-TABLE-ATTRIBUTES <table-name>)
(DRIVER-TABLE-ATTRIBUTES0 <table>) **[SUBR à 1 argument]**

Renvoie dans l'ordre de leurs créations les attributs associés à une table. Ce même ordre est utilisé pour une éventuelle création de la relation correspondante dans la base de données relationnelle.

Les fonctions DRIVER-TABLE-ATTRIBUTES[0] sont strictement identiques aux fonctions DRIVER-TABLEVAR-ATTRVARS[0].

Exemple:

```
? (DRIVER-TABLE-ATTRIBUTES 'person)
= (person.pname person.fname person.ssnum person.position person.car)
```

1.3 Description de la représentation objet

Les fonctions suivantes s'appliquent toutes au schéma de correspondances courant.
Elles permettent de décrire la représentation objet et d'ajouter au schéma des définitions de classes et de champs.

(DRIVER-DEFCLASS <class-name> <super-class-name>
<field1> ... <fieldN>)
[MACRO]

définit dans le schéma courant une classe de nom <class-name>, sous-classe de la classe de nom <super-class-name> (seul l'héritage simple est autorisé), comportant les champs <field1> ... <fieldN>. L'ordre utilisé pour décrire ces champs sera repris pour l'éventuelle création de la classe considérée.

<field1> ... <fieldN> décrivent les champs de la classe. Ce sont des listes d'au moins 3 éléments ayant la structure suivante :

(<field-name> <field-type> <specific-type-info> . <options>)

Le type du champ doit être un symbole appartenant à la liste des types possibles : (atom atomset ordatomset object object2 objectset ordobjset object2set ordobj2set monotexpr multitexpr). L'élément <specific-type-info> possède une sémantique qui varie en fonction du type du champ. (voir la fonction DRIVER-CREATE-FIELD).

Une option se présente sous la forme d'une liste à au moins deux éléments dont le premier est le nom de l'option et le ou les suivants donnent son expression ou sa valeur. Elles permettent par exemple de préciser comment calculer la valeur initiale du champ à la création de l'instance, ou encore, de contraindre un champ atomique. Voir la fonction DRIVER-CREATE-FIELD.

Il est également possible de contraindre un champ atomique ou de renforcer une contrainte posée sur un champ atomique défini dans une classe mère. <fieldM> est alors une liste à deux éléments dont le premier est le nom du champ et le second l'option contrainte, à savoir la liste composée du symbole *constraint* et de l'expression de la contrainte (lambda) telle qu'elle est spécifiée dans la description de la fonction DRIVER-CREATE-FIELD-CONSTRAINT.

Si la variable globale DRIVER-CLASS-MAKE vaut t (valeur par défaut), la fonction DRIVER-MAKE-CLASS0 est appelée et se traduit par la définition dans le langage objet client de la classe correspondante.

DRIVER-DEFCLASS peut être décrit en lisp de la manière suivante (les fonctions DRIVER-CREATE-CLASS, DRIVER-CREATE-FIELD0 et DRIVER-CREATE-CLASS-CONSTRAINT sont explicitées ci-après) :

```
(DEFMACRO DRIVER-DEFCLASS (name super-class-name . field-list)
    `(LET ((class (DRIVER-CREATE-CLASS ',name
                                       ',super-class-name t)))
          ,@(MAPCAR
               (LAMBDA (field)
                  (IF (AND (CONSP (CADR field))
                           (EQ (CAADR field) 'constraint))
                      `(DRIVER-CREATE-FIELD-CONSTRAINT ',name
                                 ',(CAR field) ',(CADADR field))
                      `(DRIVER-CREATE-FIELD0 class
                                 ,@(MAPCAR 'KWOTE field))))
               field-list)
          (IF DRIVER-CLASS-MAKE
              (DRIVER-MAKE-CLASS0 class))
          class))
```

Exemple:

```
? (DRIVER-DEFCLASS Employe ()
?            (nom      atom   symbol)
?            (prenom   atom   symbol)
```

```
?            (num-ss   atom   string)
?            (qualif   atom   symbol)
?            (chef     object Employe)
?            (salaire  atom   float)
?            (grade    atom   fix (initval 0))
?            (adresse  object Adresse)
?            (vehicule object Vehicule))
= Employe
? (DRIVER-DEFCLASS Vendeur Employe
?            (qualif (constraint
?                          (LAMBDA (qual) (EQ qual 'salesman))))
?            (situ-famille atom      symbol)
?            (commission   atom      float)
?            (sal-total    monotexpr float))
= Vendeur
? (DRIVER-DEFCLASS Departement ()
?            (nom       atom symbol)
?            (site      atom symbol (initval 'New-York)
?                       (constraint (LAMBDA (s) (MEMQ s
?                              '(New-York Washington Boston)))))
?            (chefs     ordobjset Employe)
?            (personnel objectset Employe)
?            (moy-sal   multitexpr float))
= Departement
```

DRIVER-CLASS-MAKE **[Variable]**

Si la variable globale DRIVER-CLASS-MAKE vaut t (valeur par défaut), la fonction DRIVER-MAKE-CLASS0 est exécutée à la fin de la déclaration d'une classe par DRIVER-DEFCLASS et se traduit par la définition dans le langage objet client de la classe correspondante.

(DRIVER-CREATE-CLASS <class-name>
<super-class-name> . <overwritep>)
[SUBR à 2 ou 3 arguments]

Ajoute la description de la classe de nom <class-name>, sous-classe de la classe de nom <super-class-name>, dans le schéma courant. Cette description est initialement vide de champ. Les champs correspondants devront être créés au moyen de la fonction DRIVER-CREATE-FIELD. En cas de tentative de redéfinition d'une classe, l'erreur "classe déjà existante" est provoquée, sauf si <overwritep> est précisé à t. Dans ce cas, l'ancienne définition est perdue, écrasée par la nouvelle, ainsi

que toutes les informations qu'elle contenait. Toutes les structures qui l'utilisaient pointent maintenant sur la nouvelle.

Exemple:

```
? (DRIVER-CREATE-CLASS 'Adresse ())
= Adresse
? (DRIVER-CREATE-CLASS 'Vendeur-informatique 'Vendeur)
= Vendeur-informatique
```

(DRIVER-FIND-CLASS <class-name>) **[SUBR à 1 argument]**

Retourne la description de classe de nom <class-name> si elle existe dans le schéma courant.

Exemple:

```
? (DRIVER-FIND-CLASS 'Employe)
= Employe
? (DRIVER-FIND-CLASS 'Projet)
= ()
```

(DRIVER-FIND-MAIN-CLASS <class-name>)
(DRIVER-FIND-MAIN-CLASS0 <class>) **[SUBR à 1 argument]**

Renvoie la classe principale de la classe de nom <class-name> (de l'objet <class>). Une classe principale est une classe qui n'a pas de classe mère décrite dans le schéma.

Exemple:

```
? (DRIVER-FIND-MAIN-CLASS 'Vendeur-informatique)
= Employe
```

(DRIVER-MAIN-CLASSES) **[SUBR sans argument]**

Retourne l'ensemble des classes principales décrites dans le schéma de correspondances courant.

Exemple:

```
? (DRIVER-MAIN-CLASSES)
= (Employe Departement Adresse)
```

(DRIVER-CLASS-SUBCLASSES <class-name>)
(DRIVER-CLASS-SUBCLASSES0 <class>)
(DRIVER-DIRECT-SUBCLASSES <class-name>)
(DRIVER-DIRECT-SUBCLASSES0 <class>) **[SUBR à 1 argument]**

Les fonctions DRIVER-CLASS-SUBCLASSES(0) renvoient toutes les sous-classes de la classe de nom <class-name> (resp. l'objet <class>); les fonctions DRIVER-DIRECT-SUBCLASSES(0) ne renvoient que les sous-classes directes.

Exemple:

```
? (DRIVER-CLASS-SUBCLASSES 'Employe)
= (Vendeur Vendeur-informatique)
? (DRIVER-DIRECT-SUBCLASSES 'Employe)
= (Vendeur)
```

(DRIVER-CLASS-P <class>) **[SUBR à 1 argument]**

Ce prédicat est vrai et renvoie l'objet <class> si ce dernier est une description de classe.

Exemple:

```
? (DRIVER-CLASS-P (DRIVER-FIND-CLASS 'Employe))
= Employe
```

(DRIVER-DELETE-CLASS <class-name>)
(DRIVER-DELETE-CLASS0 <class>) **[SUBR à 1 argument]**

Détruit la description de classe de nom <class-name> (resp. l'objet <class>) dans le schéma courant. Par effet de bord, les champs définis dans cette classe sont également détruits. De même, toutes les références à la classe sont retirées du schéma de correspondances.

Exemple:

```
? (DRIVER-DELETE-CLASS 'Vendeur-informatique)
= t
```

(DRIVER-CREATE-FIELD <class-name> <field-name> <ftype>
<options> . <overwritep>)

(DRIVER-CREATE-FIELD0 <class> <field-name> <ftype>

<options> . <overwritep>)
[SUBR de 3 à 6 arguments]

Ajoute la description du champ de nom <field-name> à la classe de nom <class-name> (resp. l'objet <class>) dans le schéma courant. Le type du champ doit être spécifié ainsi que certaines informations propres à chaque type. L'ensemble des types possibles est (atom atomset ordatomset object object2 objectset ordobjset object2set ordobj2set monotexpr multitexpr) :

atom Type *atomique*. Ce type caractérise le champ atomique destiné à contenir une information élémentaire. Il doit être précisé immédiatement après dans la liste des paramètres par un sous-type. Six valeurs existent par défaut; elles sont `symbol`, `string`, `fix` (ou `integer`), `float` et `()` (divers ou inconnu). Cette liste peut être complétée en fonction du modèle objet utilisé et des types qu'il reconnaît au moyen de la fonction DRIVER-ADD-ATOMIC-FIELD-TYPE. Elle est consultée à chaque création de champ atomique.

Des paramètres optionnels peuvent être précisés. Ils se présentent sous forme de listes à au moins deux éléments dont le premier est le nom de l'option et le ou les suivants donnent son expression ou sa valeur. Ces options sont :

initval Ce mot-clé introduit une forme Lisp qui va permettre de calculer la valeur initiale du champ. La forme est évaluée lors de la création de chaque instance. Attention donc de bien "coter" les constantes symboliques ou les listes qui ne sont pas des appels fonctionnels. Cette option peut être utilisée avec tous les types de champ.

constraint Cet autre mot-clé introduit une forme Lisp qui va permettre de contraindre les champs de type atomique (et uniquement les champs de ce type). L'expression est une lambda à un paramètre qui sera évaluée pour toute valeur candidate du champ (voir la description de DRIVER-CREATE-FIELD-CONSTRAINT).

localp Ce mot-clé permet de déclarer le champ comme ne faisant pas partie en fait de la classe <class-name>, mais d'une de ses classes mère. Il ne présente donc à proprement parler aucun intérêt pour la déclaration de la classe. Il permet simplement de signaler l'existence du champ <field-name> plus "haut" dans la hiérarchie, de façon à pouvoir lui définir une correspondance relationnelle au niveau de la classe <class-name> au moyen des fonctions DRIVER-DEFCLASSMAP et DRIVER-MAP-FIELD pour le cas où la classe mère le possédant ne serait pas persistante. Par défaut, un champ est considéré comme local.

atomset Type *ensemble d'atomes*. Ce type doit également être immédiatement précisé d'un argument qui définit lui le type de l'élément de l'ensemble. Les remarques effectuées pour le type atom restent valables.

ordatomset Type *ensemble ordonné d'atomes*. Ce type se rapproche du précédent. Néanmoins, l'ordre des éléments est ici important, ce qui n'était pas le cas pour le type atomset.

object Type *objet*. Ce type caractérise les champs destinés à recevoir un objet. Il doit être précisé immédiatement après dans la liste des paramètres par le nom de la classe des objets à contenir.

object2 Type *objet 2*. Il peut arriver qu'un champ objet soit amené à contenir des objets dont on ne puisse pas à l'avance connaître la classe. Cette seconde catégorie de champ objet est prévue pour ceux-là.

objectset Type *ensemble d'objets*. Ce type caractérise les champs destinés à recevoir un ensemble d'objets. Comme pour le type *objet*, il doit être immédiatement suivi par le nom de la classe commune aux objets éléments de l'ensemble.

ordobjset Type *ensemble ordonné d'objets*. Ce type se rapproche du précédent. Néanmoins, l'ordre des éléments est ici important, ce qui n'était pas le cas pour le type objectset.

object2set Type *ensemble d'objets 2*. Ce type caractérise les champs destinés à recevoir un ensemble d'objets de classes différentes et indépendantes (de classe principale différente).

ordobj2set Type *ensemble ordonné d'objets 2*. Ce type se rapproche du précédent. Néanmoins, l'ordre des éléments est ici important, ce qui n'était pas le cas pour le type object2set.

monotexpr Type *expression mono n-uplet*.

multitexpr Type *expression multi n-uplet*.

Ces deux derniers types permettent de caractériser une catégorie particulière de champ atomique. Ils ne peuvent être utilisés avec intérêt que dans la perspective du rapprochement de la représentation objet avec une représentation relationnelle. Les champs correspondants ne peuvent être consultés qu'en lecture. Ils sont destinés à recevoir le produit d'un calcul effectué dans la base de données.

Le premier des deux effectue ce calcul sur un seul n-uplet, associé généralement à l'objet considéré; Somme ou différence d'attributs du n-uplet sont des exemples d'expressions possibles.

Le second effectue le calcul sur un ensemble de n-uplets, correspondant éventuellement à un ensemble d'objets. Ce genre de calcul permet de calculer par exemple une moyenne ou la somme d'un ensemble de valeurs.

En cas de tentative de redéfinition d'un champ, l'erreur "champ déjà existant" est provoquée, sauf si <overwrite> est précisé à t. Dans ce cas, l'ancienne définition est perdue, écrasée par la nouvelle. Toutes les structures qui la contenaient pointent maintenant sur la nouvelle.

Exemple:

```
? (DRIVER-CREATE-FIELD 'Adresse 'numero 'atom 'fix)
= numero
? (DRIVER-CREATE-FIELD 'Adresse 'ville 'atom 'symbol
```

```
?                      '(initval 'New-York))
= ville
? (DRIVER-CREATE-FIELD0
?            (DRIVER-FIND-CLASS 'Adresse) 'etat 'atom 'string
?            '(constraint (LAMBDA (val)
?                            (MEMQ val '(NY MA CA)))))
= etat
? (DRIVER-CREATE-FIELD 'Employe 'vehicule2 'object 'Vehicule
?                      '(initval (OMAKEQ #:tclass:Vehicule)))
= vehicule2
```

(DRIVER-FIELD-TYPE <class-name> <field-name>) **[SUBR à 2 arguments]**

(DRIVER-FIELD-TYPE0 <field>) **[SUBR à 1 argument]**

Retourne le type du champ de nom <field-name> de la classe de nom <class-name> (resp. la classe <class-name>).

Exemple :

```
? (DRIVER-FIELD-TYPE 'Employe 'chef)
= object
```

(DRIVER-ATOMIC-FIELD-TYPES) **[SUBR sans argument]**

Retourne la liste des types atomiques de champ reconnus par DRIVER.

```
? (DRIVER-ATOMIC-FIELD-TYPES)
= (() integer fix float string symbol)
```

(DRIVER-ADD-ATOMIC-FIELD-TYPE <newtype>) **[SUBR à 1 argument]**

Définit un nouveau type de champ atomique. Cette fonction permet de compléter la liste des types utilisables disponibles dans le modèle objet utilisé.

ATTENTION: cette fonction ne vérifie pas que le modèle objet reconnaît effectivement le type rajouté.

Exemple:

```
? (DRIVER-CREATE-FIELD 'Adresse 'rue 'atom 'doublestring)
*** driver-create-field : Bad field subtype : doublestring
? (DRIVER-ADD-ATOMIC-FIELD-TYPE 'doublestring)
= doublestring
? (DRIVER-CREATE-FIELD 'Adresse 'rue 'atom 'doublestring)
= rue
```

(DRIVER-FIND-FIELD <class-name> <field-name>)
(DRIVER-FIND-FIELD0 <class> <field-name>)
(DRIVER-FIND-HERITED-FIELD <class-name> <field-name>)
(DRIVER-FIND-HERITED-FIELD0 <class> <field-name>)
(DRIVER-FIND-FIELD-IN-BRANCH <classname> <fieldname>)
(DRIVER-FIND-FIELD-IN-BRANCH0 <class> <fieldname>)
[SUBR à 2 arguments]

Les fonctions DRIVER-FIND-FIELD(0) retournent la description du champ de nom <field-name> de la classe de nom <class-name> (resp. l'objet <class>) si elle existe.

Les fonctions DRIVER-FIND-HERITED-FIELD(0) retournent la description du champ de nom <field-name> de la classe de nom <class-name> (resp. l'objet <class>) ou de l'une de ses classes mère, si cette description existe.

Les fonctions DRIVER-FIND-FIELD-IN-BRANCH(0) retournent la description du champ de nom <field-name> si elle existe dans la branche de la classe de nom <class-name> (resp. l'objet <class>). On appelle branche d'une classe l'ensemble de ses classes ancêtres jusqu'à la classe principale incluse, plus toutes ses classes descendantes (classes filles).

Exemple:

```
? (DRIVER-FIND-FIELD 'Employe 'nom)
= nom
? (DRIVER-FIND-FIELD 'Vendeur 'qualif)
= ()
? (DRIVER-FIND-HERITED-FIELD 'Vendeur 'qualif)
= qualif
? (DRIVER-FIND-FIELD-IN-BRANCH 'Employe situ-famille)
= situ-famille
```

(DRIVER-FIND-FIELD-CLASS-IN-HIERARCHY <classname> <fieldname>)
(DRIVER-FIND-FIELD-CLASS-IN-HIERARCHY0 <class> <fieldname>)
[SUBR à 2 arguments]

Recherche la classe dans laquelle se trouve le champ de nom <fieldname>. La recherche est effectuée dans la hiérarchie de la classe de nom <classname> (resp. l'objet <class>). Si plusieurs champs ont ce même nom, retourne la première classe qu'elle trouve qui possède un champ de ce nom.

Exemple :

```
? (DRIVER-FIND-FIELD-CLASS-IN-HIERARCHY 'Vendeur 'prenom)
= Employe
```

(DRIVER-FIELD-P <field>)
(DRIVER-ATOM-FIELD-P <field>)
(DRIVER-ATOMSET-FIELD-P <field>)
(DRIVER-ORDATOMSET-FIELD-P <field>)
(DRIVER-OBJECT-FIELD-P <field>)
(DRIVER-OBJECT2-FIELD-P <field>)
(DRIVER-SET-FIELD-P <field>)
(DRIVER-OBJECTSET-FIELD-P <field>)
(DRIVER-OBJECT2-FIELD-P <field>)
(DRIVER-ORDOBJSET-FIELD-P <field>)
(DRIVER-ORDOBJ2SET-FIELD-P <field>)
(DRIVER-MONOTEXPR-FIELD-P <field>)
(DRIVER-MULTITEXPR-FIELD-P <field>) **[SUBR à 1 argument]**

Le prédicat DRIVER-FIELD-P est vrai et renvoie l'objet <field> si ce dernier est un champ. Même chose pour ses pendants, avec un test portant sur les différents types de champs.

Exemple:

```
? (DRIVER-FIELD-P (DRIVER-FIND-FIELD 'Employe 'nom))
= nom
? (DRIVER-SET-FIELD-P (DRIVER-FIND-FIELD 'Employe 'nom))
= ()
```

(DRIVER-DELETE-FIELD <class-name> <field-name>)
[SUBR à 2 arguments]

(DRIVER-DELETE-FIELD0 <field>) **[SUBR à 1 argument]**

Détruit la description de champ de nom <field-name> de la classe de nom <class-name> (resp. l'objet <field>) dans le schéma courant. Par effet de bord, toute correspondance définie sur le champ est également détruite.

Exemple:

```
? (DRIVER-DELETE-FIELD 'Employe 'vehicule2)
= t
```

(DRIVER-CLASS-FIELD-P <class> <field>)
(DRIVER-HERITED-FIELD-P <class> <field>) **[SUBR à 2 arguments]**

Le prédicat DRIVER-CLASS-FIELD-P est vrai et renvoie l'objet <field> si ce dernier est une description de champ de la classe <class>. Le prédicat DRIVER-HERITED-FIELD-P est vrai et renvoie l'objet <field> si ce dernier est une description de champ de la classe <class> ou de l'une de ses classes mère.

Exemple:

```
? (DRIVER-CLASS-FIELD-P (DRIVER-FIND-CLASS 'Employe)
?                       (DRIVER-FIND-FIELD 'Employe 'prenom))
= prenom
? (DRIVER-CLASS-FIELD-P (DRIVER-FIND-CLASS 'Vendeur)
?                       (DRIVER-FIND-FIELD 'Employe 'prenom))
= ()
? (DRIVER-HERITED-FIELD-P (DRIVER-FIND-CLASS 'Vendeur)
?                         (DRIVER-FIND-FIELD 'Employe 'prenom))
= prenom
```

(DRIVER-CREATE-FIELD-CONSTRAINT <class-name> <field-name> <lambda> . <overwrite>)
[SUBR à 3 ou 4 arguments]

(DRIVER-CREATE-FIELD-CONSTRAINT0 <field> <lambda> . <overwrite>)
[SUBR à 2 ou 3 arguments]

Cette fonction permet de poser une contrainte simple sur un champ atomique.

Cette contrainte s'exprime sous la forme d'une lambda à un argument qui représente la valeur à affecter au champ. Si l'application de la lambda avec une valeur possible pour le champ renvoie vrai au sens lisp, la contrainte est respectée; elle ne l'est pas dans le cas contraire. L'expression (unique) constituant le corps de la lambda peut faire appel aux primitives lisp suivantes : $and, or, not, =, eq, neq, neqn, null, <>, <, <=, >, >=, memq$ et $nmemq$ ($(notmemq)$). Les primitives à deux arguments doivent être utilisées avec la variable comme premier argument et une constante comme second.

L'usage de l'ensemble des primitives lisp de l'interprète n'est pas autorisé car DRIVER devra pouvoir compiler l'expression en SQL.

En cas de tentative de redéfinition d'une contrainte déjà définie sur le champ dans la même classe, l'erreur "contrainte déjà existante" est provoquée, sauf si <overwrite> est précisé à t. Dans ce cas, l'ancienne définition est perdue et écrasée par la nouvelle. Toutes les descriptions qui l'utilisaient pointent maintenant sur la nouvelle.

Exemple:

```
? (DRIVER-CREATE-FIELD-CONSTRAINT 'Employe 'qualif
```

```
?                   '(LAMBDA (value)
?                        (OR (NULL value)
?                            (MEMQ value
                                  '(chairman director salesman
?                                       engineer secretary)))))
= Cstr<Employe[qualif]>
? (DRIVER-CREATE-FIELD-CONSTRAINT 'Employe 'salaire
?                   '(LAMBDA (val)
?                        (AND (>= val 1000.) (< val 10000.))))
= Cstr<Employe[salaire]>
? (DRIVER-CREATE-FIELD-CONSTRAINT 'Vendeur 'commission
?                   '(LAMBDA (comm)
?                        (OR (MEMQ comm '(100. 200. 300. 500.))
?                            (>= comm 800.))))
= Cstr<Vendeur[commission]>
```

(DRIVER-FIND-CLASS-CONSTRAINT <class-name> <field-name>)
(DRIVER-FIND-CLASS-CONSTRAINT0 <class> <field>)

[SUBR à 2 arguments]

Retourne la contrainte définie pour la classe de nom <class-name> (resp. l'objet <class>) sur le champ de nom <field-name> (resp. l'objet <field>) si elle existe.

Complément d'information sur ces deux fonctions dans la partie *Description des correspondances*.

Exemple:

```
? (DRIVER-FIND-CLASS-CONSTRAINT 'Employe 'qualif)
= Cstr<Employe[qualif]>
```

(DRIVER-FIND-FIELD-CONSTRAINTS-ON-CLASS <class-name>)
(DRIVER-FIND-FIELD-CONSTRAINTS-ON-CLASS0 <class>)

[SUBR à 1 argument]

Renvoie l'ensemble des contraintes qui s'appliquent aux champs de la classe de nom <class-name> (de l'objet <class>). Une contrainte définie sur le champ d'une classe masque l'eventuelle contrainte posée sur le même champ dans une classe mère.

Exemple:

```
? (DRIVER-FIND-FIELD-CONSTRAINTS-ON-CLASS 'Vendeur)
= (Cstr<Vendeur[qualif]> Cstr<Vendeur[commission]>
   Cstr<Employe[salaire]>)
```

(DRIVER-CLASS-CONSTRAINT-P <constraint>) **[SUBR à 1 argument]**

Ce prédicat est vrai et renvoie l'objet <constraint> si ce dernier est une contrainte de classe.

Exemple:

```
? (DRIVER-CLASS-CONSTRAINT-P
?           (DRIVER-FIND-CLASS-CONSTRAINT 'Vendeur 'qualif))
= Cstr<Vendeur[qualif]>
```

(DRIVER-DELETE-CLASS-CONSTRAINT <class-name> <field-name>)
(DRIVER-DELETE-CLASS-CONSTRAINT0 <class> <field>)
[SUBR à 2 arguments]

Supprime la contrainte définie pour la classe de nom <class-name> (resp. l'objet <class>) sur le champ de nom <field-name> (resp. l'objet <field>).

Complément d'information sur ces deux fonctions dans la partie *Description des correspondances*.

Exemple:

```
? (DRIVER-DELETE-CLASS-CONSTRAINT 'Vendeur 'commission)
= t
```

1.4 Description des correspondances

Les fonctions suivantes s'appliquent toutes au schéma de correspondances courant.

(DRIVER-DEFCLASSMAP <class-name> [<table-name>]
[<class-options>] [(fields <field1-map> ... <fieldN-map>)])

(DRIVER-DEFCLASSMAP <class-name> [<table-name>]
[<class-options>] [(letjoins (<join1> ... <joinM>)
(fields <field1-map> ... <fieldN-map>))])
[MACRO]

Définit dans le schéma courant une correspondance entre la classe de nom <class-name> et la relation <table-name>. Si la classe <class-name> est la sous-classe d'une classe dont la correspondance a déjà été précisée, <table-name> est optionnel.

Un certain nombre d'options sont disponibles pour préciser la correspondance de la classe. Une option se présente sous la forme d'une liste à au moins deux éléments dont le premier est le nom de l'option et le ou les suivants donnent son expression

ou sa valeur. Deux possibilités existent ici; elles sont *mappedp* et *attrconstraint*. Voir la description de la fonction DRIVER-MAP-CLASS.

<field1-map> ... <fieldN-map> décrivent la correspondance des champs de la classe. Ce sont des listes de 3 à 7 éléments ayant la structure suivante :

(<field-name> <field-map> <join-vars> . <options>)

Selon le type du champ (atomique, objet, ...), sa correspondance <field-map> est un attribut, l'expression d'un calcul sur la base (lambda appliquée) ou nil, complété par <join-vars> qui identifient les jointures nécessaires pour retrouver les informations.

<join-vars> est en fait une liste de noms de variables parmi celles définies au sein du *letjoins*. Il est possible de référencer une jointure définie pour la correspondance d'une classe ancêtre au moyen d'une *référence directe de jointure*. On la présente sous forme d'une liste de trois éléments, dont le premier est le nom de la classe pour laquelle la jointure a été définie et les deux autres, la table de départ et la table d'arrivée de la jointure. On ne peut pas référencer une jointure définie pour la correspondance d'une autre classe par le nom de variable qu'on lui avait associé alors car ce dernier n'a plus aucune signification en dehors du letjoins.

L'ensemble des options disponibles pour chaque type de champ est présenté dans la description de la fonction DRIVER-MAP-FIELD.

Quand la correspondance d'un champ comprend au moins une jointure, l'expression (fields <field1-map> ... <fieldN-map>) qui contient cette correspondance doit être incluse dans l'environnement

(letjoins (<join1> ... <joinM>) (fields ...))

qui permet de définir localement des jointures.

<join1> ... <joinM> sont des listes à deux éléments. le premier est un nom de variable (symbole) qui permettra de référencer la jointure dans le corps du (fields ...). Le second est une liste à trois éléments ayant la structure suivante :

(<table1-name> <table2-name> <applied-lambda>)

Pour une plus grande facilité d'usage ou par nécessité (définition d'auto-jointure), il est possible de renommer un nom de table localement à la description de la correspondance d'une classe. Il suffit de remplacer le nom de la table par la liste comprenant le nouveau nom suivi de la table (physique) dont on parle.

Consulter la description de la fonction DRIVER-CREATE-JOIN pour plus d'informations sur la création de jointure.

La description fonctionnelle de cette macro est trop complexe pour être donnée ici. DRIVER-DEFCLASS s'expanse en appels aux fonctions DRIVER-CREATE-TABLE-VAR, DRIVER-CREATE-ATTRIBUTE-VAR, DRIVER-MAP-CLASS,

DRIVER-CREATE-ATTR-CONSTRAINT, DRIVER-CREATE-JOIN et DRIVER-MAP-FIELD.

Exemple:

```
? (DRIVER-DEFCLASSMAP Employe emp
?    (attrconstraint ((LAMBDA (no) (>= no 1000))
?                     (emp empno)))
?    (LETJOINS
?        ((j1 (emp (p person)
?              ((LAMBDA (a1 a2 a3 a4)
?                  (AND (EQ a1 a3) (EQ a2 a4)))
?               (emp ename) (emp fname)
?               (p pname) (p fname))))
?         (j2 (emp (emp1 emp)
?              ((LAMBDA (a1 a2) (EQ a1 a2))
?                 (emp mgr) (emp1 empno))))
?         (j3 (emp address
?              ((LAMBDA (a1 a2) (EQ a1 a2))
?               (emp empno) (address empno))))
?         (j4 (p vehicle
?              ((LAMBDA (a1 a2) (EQ a1 a2))
?               (p car) (vehicle vnum)))))
?        (FIELDS (nom       (emp ename)    ())
?                (prenom    (emp fname)    ())
?                (num-ss    (p ssnum)      (j1))
?                (qualif    (emp job)      ())
?                (chef      ()             (j2))
?                (salaire   (emp sal)      ())
?                (adresse   ()             (j3))
?                (vehicule  ()             (j1 j4)
?                                      (filter veh-filtre))))
= t
? (DRIVER-DEFCLASSMAP Vendeur
?    (FIELDS (situ-famille (p position)      ((Employe emp p)))
            (commission   (emp com)         ())
?           (sal-total    ((LAMBDA (a1 a2) (+ a1 a2))
?                          (emp sal) (emp com)) ())))
= t
```

(DRIVER-MAP-CLASS <class-name> [<table-name>]
[<options>] . <overwritep>)

(DRIVER-MAP-CLASS0 <class> [<table>] [<options>] . <overwritep>)
[SUBR de 1 à 4 arguments]

Définit une correspondance entre classe et relation. La classe de nom <class-name> (resp. l'objet <class>) devient la représentation de la relation de nom <table-name> (resp. l'objet <table>) dans l'environnement objet, et inversement, la relation devient la correspondance relationnelle de la classe qu'on lui associe. Par défaut, tout n-uplet de la relation est un objet potentiel instance de la classe associée et tout objet persistant de la classe est un représentant dans l'environnement d'un n-uplet de la relation associée.

La relation associée à la classe est appelée sa *relation principale*. Une relation peut être choisie comme relation principale d'une classe principale que si elle n'est pas déjà relation principale d'une autre classe principale ou relation secondaire d'une autre classe. Si la classe <class-name> est la sous-classe d'une classe dont la correspondance a déjà été précisée, <table-name> n'a pas à être nécessairement re-précisé.

Les options sont toujours des listes à au moins deux éléments dont le premier est le nom de l'option et le ou les suivants donnent son expression ou sa valeur. Deux possibilités sont ici proposées, *mappedp* et *attrconstraint*:

mappedp Cette option permet de déclarer une classe comme étant sans correspondance relationnelle (alors même que l'on est en train de l'expliciter). Il y a deux intérêts à cela. D'une part, cela permet, en phase de mise au point du schéma par exemple, d'interrompre temporairement les échanges avec la base concernant une classe. Enfin, et c'est là l'intérêt majeur, cela permet de déclarer une classe sans correspondance au milieu d'une hiérarchie mappée. Toute création d'instance de la classe en question ou d'une de ses sous-classes s'effectuera ensuite correctement, les instances bénéficiant et héritant normalement des champs de la classe intermédiaire.

attrconstraint Cette option permet de poser une contrainte sur un attribut sans correspondance (non couplé à un champ atomique de la classe associée) de la relation considérée. La contrainte contribuera à filtrer les n-uplets "instances potentielles" de la classe en question qui devront nécessairement la vérifier.

Le mot-clé introduit une forme Lisp qui va permettre de contraindre un attribut. Une forme est une lambda à arguments typés, à savoir, une liste à deux éléments dont le premier est l'expression de la contrainte (lambda) telle qu'elle est spécifiée dans la description de la fonction DRIVER-CREATE-ATTR-CONSTRAINT ci-après, et le second est le "typage" de l'argument, à savoir la liste nom de la relation, nom de l'attribut dans la relation.

Exemple:

```
? (DRIVER-MAP-CLASS 'Dept 'dept)
= t
? (DRIVER-MAP-CLASS 'Adresse 'address
?     '(attrconstraint ((LAMBDA (num) (>= num 1000))
?                       (address empno))))
= t
```

(DRIVER-CLASS-MAP <class-name>)
(DRIVER-CLASS-MAP0 <class>) **[SUBR à 1 argument]**

Renvoie la table associée à la classe de nom <class-name> (resp. l'objet <class>) si elle a été spécifiée.

Exemple:

```
? (DRIVER-CLASS-MAP 'Adresse)
= address
? (DRIVER-CLASS-MAP 'Vendeur)
= emp
```

(DRIVER-TABLE-MAIN-CLASS <table-name>)
(DRIVER-TABLE-MAIN-CLASS0 <table>) **[SUBR à 1 argument]**

Renvoie la classe principale dont la correspondance relationnelle est la table de nom <table-name> (resp. l'objet <table>).

Exemple:

```
? (DRIVER-TABLE-MAIN-CLASS 'emp)
= Employe
```

(DRIVER-DELETE-CLASS-MAP <class-name>)
(DRIVER-DELETE-CLASS-MAP0 <class>) **[SUBR à 1 argument]**

Détruit la correspondance établie entre la classe principale de nom <class-name> (resp. l'objet <class>) et la table qu'on lui a associée. Cette correspondance est également détruite pour toutes ses sous-classes. Détruit également les correspondances des champs de la classe et des sous-classes.

Exemple:

```
? (DRIVER-DELETE-CLASS-MAP 'Adresse)
= t
```

(DRIVER-CREATE-ATTR-CONSTRAINT <class-name>
(<table-name> <attribute-name>) <lambda> <joins> . <overwrite>)

(DRIVER-CREATE-ATTR-CONSTRAINT0 <class> <attribute>
<lambda> <joins> . <overwrite>)
[SUBR à 4 ou 5 arguments]

Cette fonction permet de poser une contrainte simple sur un attribut. Cette contrainte s'exprime sous la forme d'une lambda à un argument où cet argument représente l'attribut à contraindre.

Exemple:

```
? (DRIVER-CREATE-ATTR-CONSTRAINT 'Adresse '(address state)
?                  '(lambda (val) (memq val '(NY CA))) ())
= Cstr<Adresse[address.state]>
```

(DRIVER-FIND-CLASS-CONSTRAINT <class-name>
(<table-name> <attribute-name>))

(DRIVER-FIND-CLASS-CONSTRAINT0 <class> <attribute>)
(DRIVER-DELETE-CLASS-CONSTRAINT <class-name>
(<table-name> <attribute-name>))

(DRIVER-DELETE-CLASS-CONSTRAINT0 <class> <attribute>)
[SUBR à 2 arguments]

Ces fonctions permettent donc également de rechercher ou détruire une contrainte posée sur un attribut. Voir les précédentes descriptions dans le paragraphe 1.3.

Exemple:

```
? (DRIVER-FIND-CLASS-CONSTRAINT 'Adresse '(address state))
= Cstr<Adresse[address.state]>
```

(DRIVER-FIND-ATTR-CONSTRAINTS-ON-CLASS <class-name>)
(DRIVER-FIND-ATTR-CONSTRAINTS-ON-CLASS0 <class>)
[SUBR à 1 argument]

Renvoie l'ensemble des contraintes s'appliquant à la classe de nom <class-name> (de l'objet <class>) et posées sur des attributs. Une contrainte définie pour une classe sur un attribut masque l'eventuelle contrainte posée sur le même attribut dans une classe mère.

Exemple:

```
? (DRIVER-FIND-ATTR-CONSTRAINTS-ON-CLASS 'Vendeur
= (Cstr<Employe[emp.empno]>)
```

(DRIVER-FIND-CONSTRAINTS-ON-CLASS <class-name>)
(DRIVER-FIND-CONSTRAINTS-ON-CLASS0 <class>)
[SUBR à 1 argument]

Renvoie l'ensemble des contraintes s'appliquant à la classe de nom <class-name> (de l'objet <class>) (champs et attributs). Une contrainte définie sur le champ d'une classe masque l'eventuelle contrainte posée sur le même champ dans une classe mère.

Exemple:

```
? (DRIVER-FIND-CONSTRAINTS-ON-CLASS 'Vendeur)
= (Cstr<Vendeur[qualif]> Cstr<Employe[salaire]> Cstr<Employe[emp.empno]>)
```

(DRIVER-CREATE-JOIN <table1-name> <table2-name> <applied-lambda> . <overwrite>)

(DRIVER-CREATE-JOIN0 <table1> <table2> <applied-lambda> . <overwrite>)
[SUBR à 3 ou 4 arguments]

Définit un chemin de la table de nom <table1-name> (resp. l'objet <table1>) vers la table de nom <table2-name> (resp. l'objet <table2>). Ce chemin peut ensuite être utilisé pour définir la correspondance d'un champ atomique de la classe de table principale <table1-name> sur un attribut de la table <table2-name>. Il peut également être utilisé pour définir la correspondance des champs objet, ensemble, et autres.

Le chemin doit être explicité sous forme d'une lambda où chaque attribut qui y participe s'exprime sous forme d'une variable passée en argument. Le chemin est établi entre deux n-uplets des deux relations quand l'application de la lambda renvoie une valeur différente de nil.

<applied-lambda> est une liste dont le premier élément est une lambda, et dont les autres représentent, sous forme de listes nom de la relation, nom de l'attribut dans la relation, l'ensemble des attributs qu'on associe à ses arguments.

Si l'application de la lambda à un n-uplet de valeurs d'attributs renvoie une valeur vraie (au sens lisp), la jointure est établie pour ce n-uplet. Elle ne l'est pas dans le cas contraire. L'expression (unique) constituant le corps de la lambda peut faire appel aux primitives lisp suivantes : $and, or, not, =, eq, neq, neqn, null, <>, <, <=, >, >=, memq$ et $nmemq$ ($(not\ memq)$). Les primitives à deux arguments doivent être utilisées avec la variable comme premier argument et une constante comme second. L'usage de l'ensemble des primitives lisp de l'interprète n'est pas autorisé car DRIVER doit ensuite pouvoir compiler l'expression en SQL pour l'exploiter.

En cas de tentative de redéfinition d'une jointure, l'erreur "jointure déjà existante" est provoquée, sauf si <overwrite> est précisé à t. Dans ce cas, l'ancienne définition est perdue, écrasée par la nouvelle. Toutes les structures qui la contenaient pointent maintenant sur la nouvelle.

Exemple:

```
? (DRIVER-CREATE-JOIN 'dept 'emp
?          '((LAMBDA (a b) (EQ a b)) (dept deptno) (emp deptn)))
= J[dept->emp]
```

(DRIVER-FIND-JOIN <table1-name> <table2-name>)
(DRIVER-FIND-JOIN0 <table1> <table2>) **[SUBR à 2 arguments]**

Retourne la jointure de la table de nom <table1-name> (resp. l'objet <table1>) vers la table de nom <table2-name> (resp. l'objet <table2> si elle existe.

Exemple:

```
? (DRIVER-FIND-JOIN 'dept 'emp)
= J[dept->emp]
```

(DRIVER-JOIN-MATCHING <table1-name> <table2-name> . <joins>)
(DRIVER-JOIN-MATCHING0 <table1> <table2> . <joins>)
[SUBR à 2 ou 3 arguments]

Retourne l'ensemble des jointures de la table de nom <table1-name> (resp. l'objet <table1>) vers la table de nom <table2-name> (resp. l'objet <table2>. L'une des deux tables (ou les deux) peut ne pas être précisée; sa référence doit alors être remplacée par ().

Si une liste de jointures <joins> est précisée, la recherche s'effectuera dedans. Sinon, l'ensemble des jointures du schéma courant sera considéré.

Exemple:

```
? (DRIVER-JOIN-MATCHING 'dept ())
= (J[dept->emp])
```

(DRIVER-JOIN-P <join>) **[SUBR à 1 argument]**

Ce prédicat est vrai et renvoie l'objet <join> si ce dernier est une jointure.

Exemple:

```
? (DRIVER-JOIN-P (DRIVER-FIND-JOIN 'dept 'emp))
= J[dept->emp]
```

(DRIVER-DELETE-JOIN <table1-name> <table2-name>)
(DRIVER-DELETE-JOIN0 <join>) **[SUBR à 1 et 2 arguments]**

Supprime la jointure de la table de nom <table1-name> vers la table de nom <table2-name> (resp. l'objet jointure <join>).

Exemple:

```
? (DRIVER-DELETE-JOIN 'dept 'emp)
= t
```

(DRIVER-MAP-FIELD <class-name> <field-name> <field-map> <join-descriptions> [<options>] . <overwritep>)
[SUBR de 4 à 8 arguments]

(DRIVER-MAP-FIELD0 <field> <field-map> <joins> [<options>] . <overwritep>)
[SUBR de 3 à 7 arguments]

Définit la correspondance du champ <field-name> (resp. l'objet <field>) de la classe <class-name> (resp. l'objet <class>) dans le monde relationnel.

En fonction du type du champ, sa correspondance peut être un attribut, une expression, un ensemble de jointures, un attribut accessible depuis la table principale par jointure.

<field-map> est la correspondance du champ, sauf si cette correspondance est un ensemble de jointures. Si la correspondance est ou comprend un ensemble de jointures, cet ensemble doit être déclaré par <join-descriptions> (resp. <joins>). <field-map> peut donc être une description d'attribut (resp. un objet attribut) ou une expression calculée (voir ci-après). Si la correspondance du champ est uniquement un ensemble de jointures, <field-map> vaut nil.

atom Les champs atomiques sont associés à un attribut. Si l'attribut fait partie de la relation principale de la classe, aucune jointure ne doit être spécifiée (<join-descriptions> (resp. <joins>) vaut nil). S'il fait partie d'une table secondaire, un ensemble de jointures reliant la table secondaire à la table principale doit être précisé.

L'option **filter** est disponible pour tous les types de champs.

filter permet de définir des filtres par lesquels passeront les données pendant les échanges avec la base. La valeur du champ lu dans la base passe dans le filtre avant écriture dans l'objet. La valeur à écrire dans la base et provenant du champ de l'objet transite également par le filtre. La définition et la description des méthodes de filtrage sont présentées au paragraphe 3.

atomset Les champs de type "ensemble d'atomes" sont associés à un attribut d'une table secondaire, relié à la relation principale par un ensemble de jointures. Cet ensemble de jointures doit permettre de sélectionner un ensemble de valeurs de l'attribut en question.

ordatomset Même chose que pour les champs de type *atomset*. En plus, l'ordre des éléments de l'ensemble est précisé par la valeur d'attributs de la même table secondaire. Le premier attribut définit l'ordre primaire, le second l'ordre secondaire, et ainsi de suite. L'option **orders** permet de préciser ces attributs et, pour chacun d'entre eux, l'ordre (ascendant ou descendant) dans lequel doit se faire le tri.

La syntaxe de l'option est la suivante :

```
(orders (attr1 ord1) ... (attrN ordN))
```

où chaque attrI est un attribut pour la fonction DRIVER-MAP-FIELD0 et une liste (<table-name> <attr-name>) pour la fonction DRIVER-MAP-FIELD, et où chaque ordI prend une valeur parmi **asc** et **desc**.

object Les champs de type "objet" représentent des liens entre objets dans la représentation objet. Ils ont naturellement comme correspondances un ensemble de jointures reliant leurs relations principales respectives. <field-map> vaut nil.

object2 Les champs de type "objet 2" retrouvent leurs valeurs en décodant une chaîne de caractères contenue dans un attribut. La correspondance de ce type de champ est donc l'attribut en question, complété de l'ensemble des jointures qui permettent de l'atteindre depuis la relation principale de la classe contenant le champ.

objectset Les champs de type "ensemble d'objets" représentent des liens entre un objet et un ensemble d'objets. Ils ont comme correspondances un ensemble de jointures reliant leurs relations principales respectives. A la différence des champs de type objet, ces jointures sont vérifiées par un ensemble de n-uplets au lieu d'un seul. <field-map> vaut nil.

ordobjset Même chose que pour les champs de type *objectset*, l'ordre des éléments de l'ensemble étant en plus précisé par la valeur d'attributs de la table secondaire. voir la description du type **ordatomset**.

object2set Les champs de type "ensemble d'objets 2" sont associés à un attribut d'une table secondaire, à partir des valeurs duquel seront décodés des pointeurs d'objet. Cet attribut doit être accessible par un ensemble de jointures. A la différence du champ de type objet2, ces jointures sont vérifiées par un ensemble de n-uplets au lieu d'un seul.

ordobj2set Même chose que pour les champs de type *ordobjset*, l'ordre des éléments de l'ensemble étant en plus précisé par la valeur d'attributs de la table secondaire. voir la description du type **ordatomset**.

monotexpr Le type "calcul sur un n-uplet" permet d'associer à un champ l'expression d'un calcul effectué à partir des n-uplets constituant l'objet, celui de la relation principale et ceux des relations secondaires. La correspondance du champ est donc un calcul s'exprimant sous forme d'une lambda appliquée à des attributs, complété de l'ensemble des jointures permettant d'accéder à ses attributs.

Sont autorisées toutes combinaisons des fonctions +, −, ∗ et /.

multitexpr Le type "calcul sur un ensemble de n-uplets" permet d'associer à un champ l'expression d'un calcul effectué sur un ensemble de n-uplets. La correspondance de ce type de champ est également une lambda appliquée, complétée de l'ensemble des jointures permettant d'accéder à l'ensemble des n-uplets en relation avec chaque objet.

Sont autorisées, en plus des fonction de calcul précédentes, les fonctions *avg* (*average* et *sum* permettant de calculer la moyenne et la somme des valeurs d'attributs accessibles par le biais de l'ensemble de jointures indiqué.

En cas de tentative de redéfinition d'une correspondance, l'erreur "correspondance déjà existante" est provoquée, sauf si <overwrite> est précisé à t. Dans ce cas, l'ancienne définition est perdue, écrasée par la nouvelle.

Exemple:

```
? (DRIVER-MAP-FIELD 'Dept 'personnel ()
?        '((dept emp ((LAMBDA (a1 a2) (EQ a1 a2))
?                     (dept deptnum) (emp empno)))))
= t
? (DRIVER-MAP-FIELD 'Dept 'moy-sal
?               '((LAMBDA (a1) (AVG a1)) (emp sal))
?               '((dept emp)))
= t
```

(DRIVER-FIELD-MAP <class-name> <field-name>)
(DRIVER-FIELD-MAP0 <field>) **[SUBR à 2 et 1 argument]**

Renvoie la correspondance du champ de nom <field-name> de la classe de nom <class-name> (resp. l'objet <field>). Cette correspondance se présente sous la forme d'une liste à deux éléments. Le premier est l'attribut (la lambda appliquée pour les champs calculés) s'il y en a un, le second est la liste des jointures, s'il y en a une.

Exemple:

```
? (DRIVER-FIELD-MAP 'Employe 'nom)
= (emp.ename ())
? (DRIVER-FIELD-MAP 'Employe 'num-ss)
= (t115@p(person).ssnum (J[emp->t115@p(person)]))
? (DRIVER-FIELD-MAP 'Employe 'voiture)
= (() (J[emp->t115@p(person)] J[t115@p(person)->t115@vehicle(vehicle)]))
```

(DRIVER-ATTRIBUTE-MAP-FIELD <table-name> <attr-name>)
[SUBR à 2 arguments]

(DRIVER-ATTRIBUTE-MAP-FIELD0 <attribute>) [SUBR à 1 argument]

Retourne le champ ayant pour correspondance l'attribut de nom <attr-name> dans la table de nom <table-name> (resp. à l'objet <table>) s'il existe.

(voir les fonctions DRIVER-ATTRIBUTE-DEF-MAP-FIELDS(0))

Exemple :

```
? (DRIVER-ATTRIBUTE-MAP-FIELD 'emp 'fname)
= prenom
? (DRIVER-ATTRIBUTE-MAP-FIELD 'person 'position)
= ()
? (DRIVER-ATTRIBUTE-MAP-FIELD 't115@p 'position)
= situ-famille
```

(DRIVER-DELETE-FIELD-MAP <class-name> <field-name>)
(DRIVER-DELETE-FIELD-MAP0 <field>) [SUBR à 2 ou 1 argument]

Détruit la correspondance établie sur le champ de nom <field-name> de la classe de nom <class-name> (resp. l'objet <field>).

Exemple:

```
? (DRIVER-DELETE-FIELD-MAP 'Dept 'personnel)
= t
```

(DRIVER-MAIN-CLASS-TABLES <class-name> . <allp>)
(DRIVER-MAIN-CLASS-TABLES0 <class> . <allp>)
[SUBR à 1 ou 2 arguments]

Retourne l'ensemble des tables élémentaires de la classe de nom <class-name> (resp. l'objet <class>) si <allp> n'est pas précisé ou vaut (), ou toutes les tables en utilisées dans la correspondance de la classe si <allp> vaut t

Exemple :

```
? (DRIVER-MAIN-CLASS-TABLES 'Employe)
= (t115@emp1(emp) emp t115@p(person))
? (DRIVER-MAIN-CLASS-TABLES 'Employe t)
= (t115@emp1(emp) t115@sg(salgrade) t115@dept(dept) t115@address(address)
  t115@vehicle(vehicle) emp t115@p(person))
```

(DRIVER-MAIN-TABLE-P <table>)
(DRIVER-SUB-TABLE-P <table>) [SUBR à 1 argument]

Le prédicat DRIVER-MAIN-TABLE-P est vrai et renvoie l'objet <table> si la table en question est table principale d'une classe. Le prédicat DRIVER-SUB-TABLE-P est vrai et renvoie également l'objet si la table en question est table secondaire d'une classe.

Une table ne peut pas être simultanément principale et secondaire. Elle peut aussi être ni l'une, ni l'autre.

Exemple:

```
? (DRIVER-MAIN-TABLE-P (DRIVER-FIND-TABLE 'emp))
= emp
? (DRIVER-SUB-TABLE-P (DRIVER-FIND-TABLE 'person))
= person
```

(DRIVER-CLASS-LOADING-ORDER <class-names>)
(DRIVER-CLASS-LOADING-ORDER0 <classes>)

[SUBR à 0 ou 1 argument]

Permet de spécifier l'ordre de chargement des classes à utiliser si des objets de différentes classes doivent être chargées au même moment. Cet ordre ne peut être quelconque : on devra forcément charger une classe avant ses filles.

Si cette fonction n'est pas employée, l'ordre de déclaration des classes est utilisé par défaut.

Si seule une partie des classes définies dans le schéma de correspondances est indiquée, ces classes seront chargées d'abord, suivies des autres selon l'ordre dans lequel elles ont été déclarées. Attention à ce que l'ordre de l'ensemble vérifie bien la restriction qui est imposée.

Renvoie l'ensemble des classes (éventuellement complété) dans l'ordre spécifié. Si aucun argument n'est passé, renvoie l'ordre courant de chargement des classes.

Exemple:

```
? (DRIVER-CLASS-LOADING-ORDER)
= (Employe Vendeur Dept Adresse)
? (DRIVER-CLASS-LOADING-ORDER '(Employe Dept Vendeur))
= (Employe Dept Vendeur Adresse)
```

(DRIVER-FIELD-LOADING-ORDER <class-name> . <field-names>)
(DRIVER-FIELD-LOADING-ORDER0 <class> . <fields>)

[SUBR à 1 ou 2 arguments]

Permet de spécifier l'ordre de chargement des champs de la classe indiquée. Si cette fonction n'est pas employée, l'ordre de déclaration des champs est utilisé par défaut.

Si seule une partie des champs définis dans la classe est indiquée, ces champs seront chargés d'abord, suivies des autres selon l'ordre dans lequel ils ont été déclarés.

Renvoie l'ensemble des champs (éventuellement complété) dans l'ordre spécifié. Si seule la classe est spécifiée, renvoie l'ordre courant de chargement de ses champs.

Exemple:

```
? (DRIVER-FIELD-LOADING-ORDER 'Employe)
= (nom prenom num-ss qualif grade chef dpt salaire adresse)
? (DRIVER-FIELD-LOADING-ORDER 'Employe
?                             '(nom prenom num-ss adresse))
= (nom prenom num-ss adresse qualif grade chef dpt salaire)
```

1.5 Description relationnelle: fonctions de bas niveau

Les fonctions présentées en 1.2 permettent de créer simplement des tables et leurs attributs pour décrire le schéma relationnel d'une base de données. Les définitions et variables de tables et d'attributs y sont gérées de manière complètement transparente pour l'utilisateur qui ne manipule en fait que des variables. Ces fonctions sont suffisantes pour l'utilisateur qui ne décrit les correspondances avec le monde objet qu'au moyen de la macro-fonction DRIVER-DEFCLASSMAP. Elles sont insuffisantes sans l'utilisation de DRIVER-DEFCLASSMAP pour exprimer les auto-jointures ou pour définir plusieurs jointures entre deux mêmes tables.

Les fonctions suivantes, de plus bas niveau, donnent accès aux objets définitions et variables de façon indépendante. Elles permettent la création de variables de tables et d'attributs directement utilisables avec les primitives de description des correspondances, en particulier pour la définition de jointure.

(DRIVER-CREATE-TABLE-DEF <table-def-name> . <overwritep>)

[SUBR à 1 ou 2 arguments]

Ajoute la définition de la table de nom <table-def-name> dans le schéma courant. Cette définition est initialement vide d'attribut. Les attributs correspondants devront être créés au moyen de la fonction DRIVER-CREATE-ATTRIBUTE-DEF.

La création d'une définition de table provoque automatiquement la création de la variable de table de même nom.

En cas de tentative de redéfinition d'une table, l'erreur "table déjà existante" est provoquée, sauf si <overwrite> est précisé à t. Dans ce cas, l'ancienne définition est perdue, écrasée par la nouvelle, ainsi que toutes les définitions d'attributs, les variables de table et d'attribut qui l'utilisaient. Toutes les structures qui le contenaient pointent maintenant sur la nouvelle.

Cette fonction ne se différencie de la fonction DRIVER-CREATE-TABLE que par le résultat qu'elle retourne qui est ici une définition de table plutôt qu'une (variable de) table dans l'autre cas.

Exemple:

```
? (DRIVER-CREATE-TABLE-DEF 'project)
= <project>
? (DRIVER-CREATE-TABLE-DEF 'car)
= <car>
```

(DRIVER-FIND-TABLE-DEF <table-def-name>) **[SUBR à 1 argument]**

Retourne la définition de table de nom <table-def-name> si elle existe dans le schéma courant.

Exemple:

```
? (DRIVER-FIND-TABLE-DEF 'project)
= <project>
```

(DRIVER-TABLE-DEF-P <table-def>) **[SUBR à 1 argument]**

Ce prédicat est vrai et renvoie l'objet <table-def> si ce dernier est une définition de table.

Exemple:

```
? (DRIVER-TABLE-DEF-P (DRIVER-FIND-TABLE-DEF 'project))
= <project>
```

(DRIVER-DELETE-TABLE-DEF <table-def-name>)
(DRIVER-DELETE-TABLE-DEF0 <table-def>) **[SUBR à 1 argument]**

Détruit la définition de table de nom <table-def-name> (resp. l'objet <table-def>) dans le schéma courant. Par effet de bord, les attributs définis dans cette table sont également détruits, ainsi que toute variable associée à la table ou à l'un de ses attributs. De même, toutes les références à la table sont retirées du schéma de correspondances.

Exemple:

```
? (DRIVER-DELETE-TABLE-DEF 'car)
= t
```

(DRIVER-CREATE-ATTRIBUTE-DEF <table-def-name>
<attr-def-name> <attr-type> <typ-len> <status> . <overwritep>)

(DRIVER-CREATE-ATTRIBUTE-DEF0 <table-def> <attr-def-name>
<attr-type> <typ-len> <status> . <overwritep>)
[SUBR à 4 ou 5 arguments]

Ajoute la définition de l'attribut de nom <attr-def-name> à la table de nom <table-def-name> (resp. à l'objet <table-def>) dans le schéma courant. Le type de l'attribut, la longueur de ce type ainsi que le statut de l'attribut doivent être spécifiés. Les différents statuts possibles sont *unique*, *not-null*, *keypart*, *void* ou *()* (voir la fonction DRIVER-CREATE-ATTRIBUTE).

Les types d'attribut possibles sont par défaut *integer*, *float* et *string*. Cette liste peut être complétée en fonction de la base de données utilisée et des types qu'elle reconnaît au moyen de la fonction DRIVER-ADD-ATTRIBUTE-TYPE. Elle est consultée à chaque création d'attribut.

La création d'une définition d'attribut provoque automatiquement la création de la variable d'attribut associée.

En cas de tentative de redéfinition d'un attribut, l'erreur "définition d'attribut déjà existante" est provoquée, sauf si <overwrite> est précisé à t. Dans ce cas, l'ancienne définition est perdue, écrasée par la nouvelle, ainsi que toutes les variables d'attribut qui l'utilisaient. Toutes les structures qui le contenaient pointent maintenant sur la nouvelle.

Ces fonctions ne se différencient des fonctions DRIVER-CREATE-ATTRIBUTE et DRIVER-CREATE-ATTRIBUTE0 que par le résultat qu'elles retournent qui est ici une définition d'attribut plutôt qu'un (une variable d') attribut dans l'autre cas.

Exemple:

```
? (DRIVER-CREATE-ATTRIBUTE-DEF 'project 'projno
?                              'integer 8 'keypart)
= <project.projno>
? (DRIVER-CREATE-ATTRIBUTE-DEF 'project 'pname 'string 20 ())
= <project.pname>
```

(DRIVER-FIND-ATTRIBUTE-DEF <table-def-name> <attr-def-name>)
(DRIVER-FIND-ATTRIBUTE-DEF0 <table-def> <attr-def-name>)
[SUBR à 2 arguments]

Retourne la définition d'attribut de nom <attr-def-name> de la table de nom <table-def-name> (de la table <table-def>) si elle existe.

Exemple:

```
? (DRIVER-FIND-ATTRIBUTE-DEF 'project 'pname)
= <project.pname>
```

(DRIVER-ATTRIBUTE-DEF-P <attr-def>) **[SUBR à 1 argument]**

Ce prédicat est vrai et renvoie l'objet <attr-def> si ce dernier est une définition d'attribut.

Exemple:

```
? (DRIVER-ATTRIBUTE-DEF-P
?           (DRIVER-FIND-ATTRIBUTE-DEF 'project 'projno))
= <project.projno>
```

(DRIVER-DELETE-ATTRIBUTE-DEF <table-def-name> <attr-def-name>)
[SUBR à 2 arguments]

(DRIVER-DELETE-ATTRIBUTE-DEF0 <attr-def>) **[SUBR à 1 argument]**

Détruit la définition d'attribut de nom <attr-def-name> de la table de nom <table-def-name> (l'objet <attr-def>) dans le schéma courant. Par effet de bord, toute variable associée à l'attribut est également détruite. De même, toutes les références à l'attribut sont retirées du schéma de correspondances.

Exemple:

```
? (DRIVER-DELETE-ATTRIBUTE-DEF 'project 'pname)
= t
```

(DRIVER-TABLEDEF-ATTRIBUTEDEFS <table-def-name>)
(DRIVER-TABLEDEF-ATTRIBUTEDEFS0 <table-def>)
[SUBR à 1 argument]

Renvoie dans l'ordre de leurs créations les définitions d'attribut associées à une définition de table. Ce même ordre est utilisé pour une éventuelle création de la relation correspondante dans la base de données relationnelle.

Exemple:

```
? (DRIVER-TABLEDEF-ATTRIBUTEDEFS 'project)
= (<project.projno>)
```

(DRIVER-ATTRIBUTE-DEF-MAP-FIELDS <table-def-name> <attr-name>)
[SUBR à 2 arguments]

(DRIVER-ATTRIBUTE-DEF-MAP-FIELDS0 <attribute-def>)
[SUBR à 1 argument]

Renvoie la liste des champs associés à une variable d'attribut dont la définition d'attribut est de nom <attr-def-name> et de table de nom <table-def-name> (resp. la table <table-def>).

Exemple :

```
? (DRIVER-ATTRIBUTE-DEF-MAP-FIELDS 'person 'position)
= (situ-famille)
```

Fonctions portant sur les variables de structure

Les variables de table et d'attribut sont utilisées pour construire les correspondances. Les définitions ne peuvent en effet en aucun cas être utilisées directement.

(DRIVER-CREATE-TABLE-VAR <table-def-name> <var-name>)
(DRIVER-CREATE-TABLE-VAR0 <table-def> <var-name>)
[SUBR à 2 arguments]

Fonctions définissant une variable associée à la définition de table de nom <table-def-name> (resp. à l'objet <table-def>) dans le schéma courant.

Exemple:

```
? (DRIVER-CREATE-TABLE-VAR 'project 'project)
** driver-create-table-var : existent table variable : project
? (DRIVER-CREATE-TABLE-VAR 'project 'proj1)
= proj1(project)
? (DRIVER-CREATE-TABLE-VAR 'project 'proj2)
= proj2(project)
```

(DRIVER-FIND-TABLE-VAR <var-name>) **[SUBR à 1 argument]**

Retourne la variable de table de nom <var-name> si elle existe dans le schéma courant.

Exemple:

```
? (DRIVER-FIND-TABLE-VAR 'project)
= project
? (DRIVER-FIND-TABLE-VAR 'proj1)
= proj1(project)
```

(DRIVER-DO-FIND-TABLE-VAR <table-def-name> <var-name>)
(DRIVER-DO-FIND-TABLE-VAR0 <table-def> <var-name>)
[SUBR à 2 arguments]

Retourne la variable de table de nom <var-name> si elle existe dans le schéma courant. Sinon, la crée associée à la définition de table de nom <table-def-name> (resp. l'objet <table-def>).

Exemple:

```
? (DRIVER-DO-FIND-TABLE-VAR 'project 'proj1)
= proj1(project)
? (DRIVER-DO-FIND-TABLE-VAR 'project 'proj2)
= proj2(project)
```

(DRIVER-TABLE-VAR-P <table-var>) **[SUBR à 1 argument]**

Ce prédicat est vrai et renvoie l'objet <table-var> si celui-ci est une variable de table.

Exemple:

```
? (DRIVER-TABLE-VAR-P (DRIVER-FIND-TABLE-VAR 'proj1))
= proj1(project)
```

(DRIVER-DELETE-TABLE-VAR <table-var-name>)
(DRIVER-DELETE-TABLE-VAR0 <table-var>) **[SUBR à 1 argument]**

Détruit la variable de table de nom <table-var-name> (resp. l'objet <table-var>) dans le schéma courant. Par effet de bord, les variables d'attributs associées à cette variable de table sont également détruites. De même, toutes les références à la table sont retirées du schéma de correspondances.

ATTENTION: Il n'est pas autorisé de détruire la variable de table automatiquement créée en même temps que la définition de table, celle portant en fait le même nom.

Exemple:

```
? (DRIVER-DELETE-TABLE-VAR 'project)
** driver-delete-table-var :
            not an erasable table variable : project
? (DRIVER-DELETE-TABLE-VAR 'proj2)
= t
```

(DRIVER-TABLEDEF-TABLEVARS <table-def-name>)
(DRIVER-TABLEDEF-TABLEVARS0 <table-def>) **[SUBR à 1 argument]**

Renvoie la liste des variables associées à la définition de table de nom <table-def-name> (resp. l'objet <table-def>).

Exemple:

```
? (DRIVER-TABLEDEF-TABLEVARS 'project)
= (project proj1(project))
```

(DRIVER-CREATE-ATTRIBUTE-VAR <tablevar-name>
<attrdef-name> . <overwritep>)

(DRIVER-CREATE-ATTRIBUTE-VAR0 <table-var>
<attribute-def> . <overwritep>)
[SUBR à 2 ou 3 arguments]

Fonctions créant une variable d'attribut associée à la variable de table de nom <tablevar-name> (resp. à l'objet <table-var>) et à la définition d'attribut de nom <attrdef-name> (resp. et à l'objet <attribute-def>).

En cas de tentative de redéfinition d'une variable d'attribut, l'erreur "variable d'attribut déjà existante" est provoquée, sauf si <overwrite> est précisé à t. Dans ce cas, il n'y a pas d'erreur. L'objet reste le même puisqu'ayant même variable de table et même définition d'attribut. L'intérêt de ce bouléen est est de pouvoir générer automatiquement des créations de variables d'attribut sans se soucier si elles existent déjà.

Exemple:

```
? (DRIVER-CREATE-ATTRIBUTE-VAR 'proj1 'projno)
= proj1(project).projno
? (DRIVER-CREATE-ATTRIBUTE-VAR 'project 'projno)
** driver-create-attribute-var :
              existent attribute variable : (project . projno)
? (DRIVER-CREATE-ATTRIBUTE-VAR0
?                   (DRIVER-FIND-TABLE-VAR 'proj1)
?                   (DRIVER-FIND-ATTRIBUTE-DEF 'Address 'city))
** driver-create-attribute-var :
                 not one of the attribute table : (project . city)
```

(DRIVER-FIND-ATTRIBUTE-VAR <table-var-name> <attr-def-name>)
(DRIVER-FIND-ATTRIBUTE-VAR0 <table-var> <attribute-def>)
[SUBR à 2 arguments]

Retourne la variable d'attribut associée à la variable de table de nom <table-var-name> (resp. à l'objet <table-var>) et à la définition d'attribut de nom <attr-def-name> (resp. et à l'objet <attribute-def>) si elle existe dans le schéma courant.

Exemple:

```
? (DRIVER-FIND-ATTRIBUTE-VAR 'project 'projno)
= project.projno
? (DRIVER-FIND-ATTRIBUTE-VAR 'proj1 'projno)
= proj1(project).empno
```

(DRIVER-DO-FIND-ATTRIBUTE-VAR <table-var-name> <attr-def-name>)
(DRIVER-DO-FIND-ATTRIBUTE-VAR0 <table-var> <attribute-def>)
[SUBR à 2 arguments]

Retourne la variable d'attribut associée à la variable de table de nom <table-var-name> (resp. à l'objet <table-var>) et à la définition d'attribut de nom <attr-def-name> (resp. et à l'objet <attribute-def>) si elle existe dans le schéma courant. Sinon, la crée et la renvoie.

Exemple:

```
? (DRIVER-DO-FIND-ATTRIBUTE-VAR 'proj1 'projno)
= proj1(project).projno
? (DRIVER-DO-FIND-ATTRIBUTE-VAR 'proj2 'projno)
= proj2(project).empno
```

(DRIVER-ATTRIBUTE-VAR-P <attribute-var>) **[SUBR à 1 argument]**

Ce prédicat est vrai et renvoie l'objet <attribute-var> si celui-ci est une variable d'attribut.

Exemple:

```
? (DRIVER-ATTRIBUTE-VAR-P
?           (DRIVER-FIND-ATTRIBUTE-VAR 'proj1 'projno))
= proj1(project).projno
```

(DRIVER-ATTRDEF-ATTRVARS <table-def-name> <attribute-def-name>)
[SUBR à 2 arguments]

(DRIVER-ATTRDEF-ATTRVARS0 <attribute-def>) **[SUBR à 1 argument]**

Renvoie la liste des variables associées à la définition d'attribut de nom <attribute-def-name> de la table de nom <table-def-name> (resp. à l'objet <attribute-def> pour la seconde fonction).

Exemple:

```
? (DRIVER-ATTRDEF-ATTRVARS 'project 'projno)
= (project.projno proj1(project).projno)
```

(DRIVER-TABLEVAR-ATTRVARS <table-var-name>)
(DRIVER-TABLEVAR-ATTRVARS0 <table-var>) **[SUBR à 1 argument]**

Renvoie la liste des variables d'attribut définies à partir de la variable de table de nom <table-var-name> (resp. l'objet <table-var>).

Exemple:

```
? (DRIVER-TABLEVAR-ATTRVARS 'proj1)
= (proj1(project).projno)
```

(DRIVER-DELETE-ATTRIBUTE-VAR <table-var-name> <attr-def-name>)
(DRIVER-DELETE-ATTRIBUTE-VAR0 <attribute-var>)

[SUBR à 1 argument]

Détruit la variable d'attribut associée à la variable de table de nom <table-var-name> et à la définition d'attribut de nom <attr-def-name> (resp. l'objet <attribute-var>) si elle existe dans le schéma courant. Par effet de bord, toutes les références à l'attribut sont retirées du schéma de correspondances. Par exemple, l'apparition de la variable dans une jointure provoquera la destruction de la jointure.

ATTENTION: Il n'est pas autorisé de détruire la variable d'attribut automatiquement créée en même temps que la définition d'attribut, celle dont la variable de table associée porte le même nom que la définition de table correspondante.

Exemple:

```
? (DRIVER-DELETE-ATTRIBUTE-VAR 'proj1 'projno)
= t
? (DRIVER-DELETE-ATTRIBUTE-VAR 'project 'projno)
** driver-delete-table-var :
            not an erasable attribute variable : (project projno)
```

1.6 Création des classes et des relations

Ces fonctions permettent de définir les structures objet et relationnelles respectivement dans le langage objet et dans le SGBD connecté si elles n'y existent pas encore.

(DRIVER-MAKE-TABLE <table-name>)
(DRIVER-MAKE-TABLE0 <table>) **[SUBR à 1 argument]**

Définit dans le SGBD associé au schéma de correspondances courant la table de nom <table-name> (resp. l'objet <table>) précédemment décrite dans ce même schéma.

ATTENTION: Cette fonction ne vérifie pas l'inexistence de la table en question dans le SGBD avant la création demandée.

Exemple:

```
? (DRIVER-MAKE-TABLE 'emp)
= t
```

(DRIVER-MAKE-CLASS <class-name>)
(DRIVER-MAKE-CLASS0 <class>) **[SUBR à 1 argument]**

Définit dans le langage objet hôte la classe de nom <class-name> précédemment décrite dans le schéma de correspondances courant.

Exemple:

```
? (DRIVER-MAKE-CLASS 'Employe)
= t
```

1.7 Génération automatique de classes et de relations

Ces fonctions permettent de générer automatiquement, à partir d'une représentation objet ou relationnelle, celle qui lui correspond dans l'autre modèle. Les rapprochements sont également produits.

(DRIVER-GENERATE-TABLE-MAP <table-name>) **[MACRO]**

Définit dans le schéma courant une correspondance objet pour la table de nom <table-name>. La classe générée comporte un champ atomique par attribut, le type de l'attribut étant repris comme type du champ. Il conviendra donc de définir les méthodes de filtrage en lecture - écriture des champs avec la base pour les attributs ayant un type différent d'integer, float et string.

DRIVER-GENERATE-TABLE-MAP peut être décrit en lisp de la manière suivante :

```
(DEFMACRO DRIVER-GENERATE-TABLE-MAP (table-name)
   (LET ((table-def (SEND 'table
                          (DRIVER-FIND-TABLE table-name))))
    `(PROGN
       (DRIVER-DEFCLASS ,(SEND 'name table-def) ()
                ,@(MAPCAR (LAMBDA (attr-def)
                            `(,(SEND 'attrname attr-def)
                              atom
                              ,(SEND 'type attr-def)))
                          (DRIVER-TABLEDEF-ATTRIBUTEDEFS0
                                                table-def)))
       (DRIVER-DEFCLASSMAP ,(SEND 'name table-def)
                           ,(SEND 'name table-def)
                (FIELDS
                   ,@(MAPCAR (LAMBDA (attr-def)
                               `(,(SEND 'attrname attr-def)
                                 (,(SEND 'name table-def)
                                  ,(SEND 'attrname attr-def))
                                 ()))
                             (DRIVER-TABLEDEF-ATTRIBUTEDEFS0
                                              table-def)))))))
```

Exemple:

```
? (DE essai () (DRIVER-GENERATE-TABLE-MAP emp))
= essai
? ^Pessai
(de essai ()
    (driver-generate-table-map emp))
= ()
? (essai)
** driver-map-class : Yet a main table : emp
? ; emp est deja table principale de la classe Employe !
? (driver-delete-class-map 'Employe)
= t
? (essai)
= t
? ^Pessai
(de essai ()
    (progn
      (let ((class (driver-create-class 'emp '() t)))
           (driver-create-field0 class 'empno 'atom 'integer)
           (driver-create-field0 class 'ename 'atom 'string)
           (driver-create-field0 class 'fname 'atom 'string)
```

```
              (driver-create-field0 class 'job 'atom 'string)
              (driver-create-field0 class 'mgr 'atom 'integer)
              (driver-create-field0 class 'sal 'atom 'float)
              (driver-create-field0 class 'com 'atom 'float)
              (driver-create-field0 class 'deptn 'atom 'integer)
              (if driver-class-make (driver-make-class0 class))
              class)
              (progn
                (driver-map-class 'emp 'emp t)
                (driver-map-field 'emp 'empno '(emp empno) '())
                (driver-map-field 'emp 'ename '(emp ename) '())
                (driver-map-field 'emp 'fname '(emp fname) '())
                (driver-map-field 'emp 'job '(emp job) '())
                (driver-map-field 'emp 'mgr '(emp mgr) '())
                (driver-map-field 'emp 'sal '(emp sal) '())
                (driver-map-field 'emp 'com '(emp com) '())
                (driver-map-field 'emp 'deptn '(emp deptn) '())
                t)))
= ()
```

(DRIVER-GENERATE-CLASS-MAP <class-name>) **[MACRO]**

Définit dans le schéma courant une correspondance relationnelle pour la class de nom <class-name>.

DRIVER-GENERATE-CLASS-MAP n'est pas encore implémentée dans cette version de DRIVER.

2 Les primitives utilisateur

2.1 Connexion et communication avec le SGBD

(DRIVER-CONNECT <dbms-name> <base-name>) [SUBR à 2 arguments]

Assure la connexion à la base de données de nom <base-name> définie dans le SGBD de nom <dbms-name>. Retourne le curseur résultant de cette connexion.

À la première utilisation de DRIVER-CONNECT lors d'une session, le curseur est stocké dans une variable système. Il est utilisé par défaut comme curseur courant dans tout schéma de correspondances ultérieurement créé au moyen de la fonction DRIVER-CREATE-MAPPING si la variable système DRIVER-AFFECT-CURSOR-P vaut t (valeur par défaut).

Après avoir établi la connexion et créé le curseur, DRIVER-CONNECT effectue un appel à la fonction DRIVER-DBMS-CONNECTION avec les deux arguments <dbms-name> et <base-name> avant de rendre la main.

Exemple de connexion sur la base `smecidemo` avec le SGBD ingres :

```
? (DRIVER-CONNECT 'ingres 'smecidemo)
= #:tclass:cursor:#[ingres 0 t ()]
```

(DRIVER-DBMS-CONNECTION <dbms-name> <user-name>)
[SUBR à 2 arguments]

Fonction appelée par DRIVER-CONNECT lors de la connexion à une base de données. Par défaut, cette fonction est vide et a comme définition :

```
(de driver-dbms-connection (dbms-name user-name))
```

L'utilisateur a tout loisir de la redéfinir pour ses besoins propres.

(DRIVER-CREATE-NEW-CURSOR <dbms-name> <base-name>)
(DRIVER-CREATE-NEW-CURSOR0 <dbms>) [SUBR à 1 ou 2 arguments]

Crée et renvoie un nouveau curseur sur la base de données de nom <base-name> définie dans le SGBD de nom <dbms-name>.

Exemple:

```
? (setq c2 (DRIVER-CREATE-NEW-CURSOR 'ingres 'smecidemo))
= #:tclass:cursor:#[ingres 0 t ()]
```

(DRIVER-MAPPING-CURSOR <mapping-name> . <cursor>)
(DRIVER-MAPPING-CURSOR0 <mapping> . <cursor>)
[SUBR à 1 ou 2 arguments]

Variables-fonctions permettant d'accéder en lecture au curseur courant du schéma de correspondances de nom <mapping-name> si <cursor> n'est pas fourni, et de le remplacer par <cursor> dans le cas contraire.

Exemple:

```
? (DRIVER-MAPPING-CURSOR 'map1)
= #:tclass:cursor:#[ingres 0 t ()]
```

(DRIVER-CURRENT-CURSOR <cursor>) **[SUBR à 0 ou 1 argument]**

Variable-fonction permettant d'accéder en lecture au curseur courant du schéma courant si <cursor> n'est pas fourni, et de le remplacer par <cursor> dans le cas contraire.

REMARQUE : Un curseur n'est courant que relativement à un schéma donné. Si le schéma courant change, le curseur courant est également susceptible de changer.

Exemple:

```
? (DRIVER-CURRENT-CURSOR)
= #:tclass:cursor:#[ingres 0 t ()]
```

DRIVER-FATAL-REQUEST-ERROR **[Variable]**

Si la variable système DRIVER-FATAL-REQUEST-ERROR vaut t (valeur par défaut), tout échec de requête pour cause d'erreur détectée par le SGBD provoque une erreur DRIVER. Si elle vaut (), l'exécution continue comme si la requête était valide.

2.2 Filtrage et manipulation des objets relationnels

DRIVER-COMMIT-AFTER-READING **[Variable]**

Si la variable système DRIVER-COMMIT-AFTER-READING vaut t (valeur par défaut), toutes les requêtes de consultation qui ne sont pas faites dans le cadre d'écritures d'objets dans la base sont suivies d'une validation SGBD (commit SQL) qui déverrouille les données et les rend accessibles aux autres utilisateurs. Dans le cas contraire, seul DRIVER-COMMIT-OBJECTS déclenche une validation SGBD.

(DRIVER-EXISTENT-KEY-P <class-name> <key-value>)
(DRIVER-EXISTENT-KEY-P0 <class> <key-value>)

[SUBR à 2 arguments]

Ce prédicat est vrai et renvoie t si la valeur de clé <key-value> (liste de valeurs correspondant à la liste des attributs composant la clé de la table principale associée à la classe) correspond à un objet relationnel au moins instance de la classe de nom <class-name> (resp. l'objet <class>).

Cette fonction n'est utilisable que quand la connexion à la base a été réalisée car elle engendre une requête SQL qui est immédiatement envoyée à la base pour tester l'existence d'un n-uplet relationnel.

Exemple:

Les réponses retournées par DRIVER-EXISTENT-KEY-P ci-après ont été obtenues alors que :

- le tuple de clé 50 dans la relation emp n'existe pas;
- le tuple de clé 7027 existe mais ne vérifie pas les contraintes imposées par la classe Employé;
- le tuple de clé 7698 existe et vérifie les contraintes imposées par la classe Employé sans vérifier celles imposées par la classe Vendeur;
- le tuple de clé 7499 existe et vérifie les contraintes imposées par la classe Vendeur.

```
? (DRIVER-EXISTENT-KEY-P 'Employe '(50))
= ()
? (DRIVER-EXISTENT-KEY-P 'Employe '(7027))
= ()
? (DRIVER-EXISTENT-KEY-P 'Employe '(7698))
= t
? (DRIVER-EXISTENT-KEY-P 'Vendeur '(7698))
= ()
? (DRIVER-EXISTENT-KEY-P 'Employe '(7499))
= t
? (DRIVER-EXISTENT-KEY-P 'Vendeur '(7499))
= t
```

(DRIVER-GET-OBJECT <class-name> <key-value>)
(DRIVER-GET-OBJECT0 <class> <key-value>) **[SUBR à 2 arguments]**

Renvoie l'objet jumeau associé à l'objet relationnel de classe de nom <class-name> (resp. l'objet <class>) dont la clé dans la base a pour valeur <key-value> (liste de valeurs correspondant à la liste des attributs composant la clé de la table principale associée à la classe), s'il a été construit, et sinon, le défaut d'objet le représentant.

Si la variable système DRIVER-EXISTENT-KEY-TEST vaut t (la valeur par défaut est nil), DRIVER vérifie que la valeur de clé existe.

Exemple:

```
? (DRIVER-GET-OBJECT 'Employe '(7499))
= driver-object-default-1
```

DRIVER-EXISTENT-KEY-TEST **[Variable]**

Si la variable système DRIVER-EXISTENT-KEY-TEST vaut t (la valeur par défaut est nil), DRIVER vérifie à l'appel de DRIVER-GET-OBJECT que la valeur de clé correspond effectivement à un objet relationnel en appelant la fonction DRIVER-EXISTENT-KEY-P. Cette option est relativement coûteuse puisqu'elle engendre une requête SQL qui est immédiatement envoyée à la base.

Exemple :

```
? DRIVER-EXISTENT-KEY-TEST
= ()
? (DRIVER-GET-OBJECT 'Employe '(1)) ; clef inexistante dans la base
= driver-object-default-2
? (SETQ DRIVER-EXISTENT-KEY-TEST t)
= t
? (DRIVER-GET-OBJECT 'Employe '(1))
** driver-get-object : Not an existent object key : (Employe 1)
```

(DRIVER-GET-OBJECT-IF-LOADED <class-name> <key-value>)
(DRIVER-GET-OBJECT-IF-LOADED0 <class> <key-value>)
[SUBR à 2 arguments]

Renvoie l'objet jumeau associé à l'objet relationnel de classe de nom <class-name> (resp. l'objet <class>) et de valeur de clé <key-value> (liste de valeurs correspondant à la liste des attributs composant la clé de la table principale associée à la classe), s'il a été construit.

Exemple:

```
? (DRIVER-GET-OBJECT-IF-LOADED 'Employe '(7499))
= ()
```

(DRIVER-SELECT-OBJECTS <applied-lambda>) **[SUBR à 1 argument]**

Renvoie la liste des ensembles de valeurs de clé correspondant aux ensembles d'objets qui, dans la base, vérifient le filtre exprimé sous la forme d'une lambda appliquée <applied-lambda> :

((LAMBDA (obj1 ... objN) corps-lambda) classe-obj1 ... classe-objN)

<applied-lambda> est une liste dont le premier élément est une lambda, et dont les autres précisent la classe des objets associés à ses arguments. Les arguments de la lambda seront chacun instanciés à un des objets des n-uplets à sélectionner.

Un n-uplet d'objets vérifie le filtre si l'application de la lambda à ce n-uplet renvoie une valeur vraie au sens lisp.

Le corps de la lambda doit être constituée d'une expression unique pouvant faire appel aux primitives lisp suivantes : $and, or, not, =, eq, neq, neqn, null, <>, <, <=, >, >=, memq$ et $nmemq$ ($(notmemq)$). Les primitives à deux arguments doivent être utilisées avec la variable comme premier argument et une constante comme second. Comme les variables représentent des objets, la fonction SEND peut en plus être utilisée pour accéder en lecture aux contenus de leurs champs.

De même, comme DRIVER-SELECT-OBJECTS peut en principe servir dans un environnement dynamique, les variables définies avant l'appel de la fonction (par LET ou DEFVAR par exemple) peuvent être utilisées dans le corps de la lambda.

Restrictions

L'accès aux champs de type calcul multi-tuples **multitexpr** n'est pas autorisé pour définir le filtre dans la version actuelle de cette fonction.

D'autre part, les méthodes *driverloading* destinées à modifier une valeur issue de la base avant de l'affecter dans un champ ne sont pas utilisées puisque le filtre est directement compilé en SQL. Seules les valeurs brutes des attributs seront donc exploitées.

Enfin, l'usage d'autres primitives lisp de l'interprète n'est pas autorisé car DRIVER devra pouvoir compiler l'expression en SQL.

Exemple:

```
? ; Filtrer les employes dont le salaire est superieur au seuil
? (DRIVER-SELECT-OBJECTS
?    '((LAMBDA (o1) (>= (SEND 'salaire o1) seuil))
?      Employe))
*** driver-select-objects : unbounded variable : seuil
? (LET ((seuil 1500.))
?      (DRIVER-SELECT-OBJECTS
```

```
?         '((LAMBDA (o1) (>= (SEND 'salaire o1) seuil))
?           Employe)))
= (((7839)) ((7566)) ((7698)) ((7782))
   ((7902)) ((7788)) ((7499)))
?
? ; Filtrer les employe ayant une voiture de meme modele
? ; que leur chef
? (DRIVER-SELECT-OBJECTS
?    '((LAMBDA (obj)
?         (EQ (SEND 'modele (SEND 'vehicule obj))
?             (SEND 'modele (SEND 'vehicule (SEND 'chef obj)))))
?      Employe))
= (((7782)))
?
? ; Filtrer les couples d'employes ayant meme chef et dont le
? ; salaire du premier est superieur a celui du second
? (DRIVER-SELECT-OBJECTS
?    '((LAMBDA (o1 o2)
?        (AND (EQ (SEND 'chef o1) (SEND 'chef o2))
?             (> (SEND 'salaire o1) (SEND 'salaire o2))))
?      Employe Employe))
= (((7655) (7521)) ((7655) (7654)) ((7499) (7521))
   ((7499) (7654)) ((7499) (7655)) ((7698) (7782))
   ((7566) (7782)) ((7566) (7698)))
? ; Filtrer les couples Vendeur-Employe, tous deux ayant meme
? ; chef, tel que le salaire du vendeur est superieur a celui
? ; de l'employe
? (DRIVER-SELECT-OBJECTS
?    '((LAMBDA (o1 o2)
?        (AND (EQ (SEND 'chef o1) (SEND 'chef o2))
?             (> (SEND 'salaire o1) (SEND 'salaire o2))))
?      Vendeur Employe))
= (((7499) (7521)) ((7499) (7654)) ((7499) (7655))
   ((7655) (7521)) ((7655) (7654)))
```

(DRIVER-LOAD-CLASS-OBJECTS <class-name> . <key-values>)
(DRIVER-LOAD-CLASS-OBJECTS0 <class> . <key-values>)
[SUBR à 1 ou 2 arguments]

Construit et renvoie les objets jumeaux associés aux objets relationnels de classe principale de nom <class-name> (resp. l'objet <class>) dont les clés dans la base ont pour valeur <key-values> (liste des listes de valeurs correspondant à la liste des attributs composant la clé de la table principale associée à la classe). Si <class-name> est le nom d'une classe qui n'est pas principale, sa classe principale est

automatiquement considérée. Si <key-values> vaut t, tous les objets jumeaux correspondant à la classe principale précisée sont construits.

Quand un objet existe déjà dans l'environnement, le contenu de ses champs est perdu, écrasé par les valeurs issues de la base. De même, sa classe est éventuellement mise à jour.

Exemple:

```
? (DRIVER-LOAD-CLASS-OBJECTS 'Employe '((7499)))
= (allen)
? (DRIVER-GET-OBJECT-IF-LOADED 'Employe '(7499))
= allen
? (DRIVER-LOAD-CLASS-OBJECTS 'Departement t)
= (accounting research sales)
```

(DRIVER-OBJECT-LOADING-BEGIN <main-class>) **[SUBR à 1 argument]**

Cette fonction est appelée par DRIVER avant tout chargement en mémoire d'un groupe d'objets. La classe principale des objets est passé en argument. Par défaut, elle ne fait rien, sa définition est :

```
(de driver-object-loading-begin (mainclass))
```

Elle peut bien sûr être redéfinie par l'utilisateur.

(DRIVER-OBJECT-LOADING-END <main-class>) **[SUBR à 1 argument]**

Cette fonction est appelée par DRIVER après tout chargement en mémoire d'un groupe d'objets. La classe principale des objets est passé en argument. Par défaut, elle ne fait rien, sa définition est :

```
(de driver-object-loading-end (mainclass))
```

Elle peut bien sûr être redéfinie par l'utilisateur.

(DRIVER-OBJECT-LOADED <object> <class>) **[SUBR à 2 arguments]**

DRIVER envoie le message driver-object-loaded avec l'argument <class> à tout objet jumeau dont les champs propres à la classe <class> viennent d'être chargés.

L'utilisateur a donc tout loisir de définir des méthodes pour les classes persistantes du schéma. La méthode appelée par défaut est la fonction DRIVER-OBJECT-LOADED qui ne fait rien. DRIVER-OBJECT-LOADED peut bien sûr être redéfinie.

Exemple de méthode avec le modèle objet MicroCeyx :

```
(de {Employe}:driver-object-loaded (object class)
    (print "Employe " object
           "charge (Champs propres a la classe " class ")"))
```

(DRIVER-LOAD-DEEP-OBJECTS <objects>) **[SUBR à 1 argument]**

Construit les objets jumeaux des objets passés en arguments et tous les objets qu'ils référencent directement ou indirectement.

ATTENTION : Seuls les objets passés en argument sont rechargés (anciennes valeurs écrasées). Les objets référencés qui ont déjà été construits dans l'environnement ne sont pas rechargés mais seulement retrouvés en mémoire.

Exemple:

```
? (DRIVER-LOAD-DEEP-OBJECTS (LIST (DRIVER-GET-OBJECT 'Adresse '(7499))))
= (adresse-23)
```

(DRIVER-ALL-PERSISTENT-OBJECTS-IN-MEMORY) [SUBR sans argument]

Retourne la liste de tous les objets persistants référencés dans l'environnement, que ce soient des objets jumeaux ou des défauts d'objet.

Exemple :

```
? (DRIVER-ALL-PERSISTENT-OBJECTS-IN-MEMORY)
= ()
```

2.3 Manipulation explicite des défauts d'objet

Les fonctions DRIVER-GET-OBJECT(0) permettent de contruire le défaut d'objet d'un objet relationnel qui n'aurait pas encore de référence dans l'environnement.

(DRIVER-OBJECT-DEFAULT-CLASS) **[SUBR sans argument]**

Renvoie l'objet classe DRIVER des défauts d'objet. La construit si elle n'existe pas déjà.

Exemple:

```
? (DRIVER-OBJECT-DEFAULT-CLASS)
= driver-object-default
```

(DRIVER-OBJECT-DEFAULT-P <object-default>) [SUBR à 1 argument]

Ce prédicat est vrai et renvoie l'objet <object-default> si ce dernier est un défaut d'objet.

Exemple:

```
? (DRIVER-OBJECT-DEFAULT-P (DRIVER-GET-OBJECT 'Employe '(7698)))
= driver-object-default-17
```

(DRIVER-LOAD-OBJECT-DEFAULT <object-default>)

[SUBR à 1 argument]

Construit et renvoie l'objet jumeau associé au défaut d'objet <object-default>. L'objet jumeau se substitue au défaut d'objet partout où ce dernier était référencé. Si le défaut d'objet correspond à un objet qui est manquant, la fonction DRIVER-MISSING-OBJECT-PROCESSING est appelée.

Exemple:

```
? (DRIVER-LOAD-OBJECT-DEFAULT
?           (DRIVER-GET-OBJECT 'Employe '(7698)))
= blake
```

(DRIVER-MISSING-OBJECT-PROCESSING <default>)

[SUBR à 1 argument]

En cas de tentative malheureuse de chargement d'un défaut d'objet, la fonction DRIVER-MISSING-OBJECT-PROCESSING est appelée avec, en argument, le défaut d'objet incriminé. Par défaut, elle déclenche une erreur "Objet manquant" :

```
(de DRIVER-MISSING-OBJECT-PROCESSING (default)
    (error 'driver-missing-object-processing
           "Missing relational object" default))
```

Elle peut être redéfinie.

2.4 Accès à l'objet relationnel

(DRIVER-SEND <message> <object> . <args>)
[SUBR à 2 et plus arguments]

Envoie le message <message> à l'objet <object> avec les arguments <args>. Si <object> est un défaut d'objet, le jumeau est construit et se substitue à lui avant l'envoi de message.

Exemple :

```
? (DRIVER-GET-OBJECT 'Employe '(7839))
= driver-object-default-30
? (DRIVER-SEND 'salaire (DRIVER-GET-OBJECT 'Employe '(7839)))
= 5000.
? (DRIVER-GET-OBJECT 'Employe '(7839))
= king
```

(DRIVER-READ-FIELD-VALUE <field-name> <object>)
[SUBR à 2 arguments]

Renvoie le contenu du champ <field-name> de l'objet <object>. Si ce dernier est encore un défaut d'objet, le jumeau est construit et se substitue à lui avant lecture du champ.

Exemple:

```
? (DRIVER-GET-OBJECT 'Employe '(7654))
= driver-object-default-18
? (DRIVER-READ-FIELD-VALUE 'salaire
?                (DRIVER-GET-OBJECT 'Employe '(7654)))
= 1250
? (DRIVER-GET-OBJECT 'Employe '(7654))
= martin
```

(DRIVER-SET-FIELD-VALUE <field-name> <object> <value>)
[SUBR à 3 arguments]

Affecte le champ <field-name> de l'objet <object> avec la valeur <value>. Si ce dernier est encore un défaut d'objet, le jumeau est construit et se substitue à lui avant écriture du champ.

Exemple:

```
? (DRIVER-GET-OBJECT 'Employe '(7329))
= driver-object-default-14
? (DRIVER-SET-FIELD-VALUE 'salaire
?               (DRIVER-GET-OBJECT 'Employe '(7329)) 1500.)
= 1500.
? (DRIVER-GET-OBJECT 'Employe '(7329))
= smith
```

DRIVER-UPDATE-DAEMON-P **[Variable]**

Si la variable système DRIVER-UPDATE-DAEMON-P vaut t (la valeur par défaut est ()), DRIVER met à jour un objet jumeau s'il a connaissance par l'une des fonctions DRIVER-READ-FIELD-VALUE-IN-DATABASE(0) ou DRIVER-SET-FIELD-VALUE-IN-DATABASE(0) que l'objet relationnel correspondant a une valeur différente.

Pour plus de détails, consulter la description de ces fonctions.

(DRIVER-READ-FIELD-VALUE-IN-DATABASE <field-name> <object>)
(DRIVER-READ-FIELD-VALUE-IN-DATABASE0 <field> <object>)
[SUBR à 2 arguments]

Renvoie le contenu du champ <field-name> (resp. l'objet champ <field>) de l'objet <object> en allant directement lire la valeur dans la base de données. La lecture du champ ne provoque pas la mutation d'un défaut d'objet en jumeau.

Quand la variable DRIVER-UPDATE-DAEMON-P vaut t (la valeur par défaut est nil), si l'objet jumeau est déjà constitué et que la valeur du champ lue dans la base est différente de celle se trouvant dans l'objet, le champ en question est mis à jour dans l'objet jumeau.

Exemple:

```
? (DRIVER-GET-OBJECT 'Employe '(7655))
= driver-object-default-19
? (DRIVER-READ-FIELD-VALUE-IN-DATABASE 'salaire
?               (DRIVER-GET-OBJECT 'Employe '(7655)))
= 1350
? (DRIVER-GET-OBJECT 'Employe '(7655))
= driver-object-default-19
```

(DRIVER-SET-FIELD-VALUE-IN-DATABASE <field-name>
<object> <value>)

(DRIVER-SET-FIELD-VALUE-IN-DATABASE0 <field> <object> <value>)
[SUBR à 3 arguments]

Affecte le champ <field-name> (resp. l'objet champ <field>) de l'objet <object> avec la valeur <value> directement dans la base de données. L'écriture du champ ne provoque pas la mutation d'un défaut d'objet en jumeau.

On ne vérifie pas que la valeur affectée dans la base respecte les constraintes de classe de l'objet. Ainsi, une affectation dans la base peut provoquer la mutation d'un objet vers une nouvelle classe. La classe de l'objet relationnel devient alors différente de la classe de l'objet jumeau correspondant.

Quand la variable DRIVER-UPDATE-DAEMON-P vaut t (la valeur par défaut est nil), si l'objet jumeau est déjà constitué et que la valeur du champ écrite dans la base est différente de celle se trouvant dans l'objet, le champ en question est mis à jour dans l'objet jumeau.

Quand la variable DRIVER-MESSAGES-P vaut t (valeur par défaut), un message est imprimé dans le terminal quand DRIVER modifie la base de données.

ATTENTION Ces fonctions permettent d'aller directement écrire dans la base une information. Elles violent donc les règles qui régissent les transactions DRIVER. Leur utilisation doit donc être circonspecte et pertinente.

Exemple:

```
? (DRIVER-GET-OBJECT 'Employe '(7655))
= driver-object-default-19
? (DRIVER-SET-FIELD-VALUE-IN-DATABASE 'salaire
?               (DRIVER-GET-OBJECT 'Employe '(7655)) 1400.)
Mise-a-jour de l'objet driver-object-default-19
= 1400.
? (DRIVER-GET-OBJECT 'Employe '(7655))
= driver-object-default-19
? (DRIVER-READ-FIELD-VALUE-IN-DATABASE 'salaire
?                (DRIVER-GET-OBJECT 'Employe '(7655)))
= 1400.
```

(DRIVER-OBJECT-CLASS-IN-DATABASE <object>)
[SUBR à 1 argument]

Renvoie la classe que l'objet <object> a dans la base de données. Cette classe peut être différente de celle de l'objet <object>.

Exemple:

```
? (DRIVER-OBJECT-CLASS-IN-DATABASE (DRIVER-GET-OBJECT 'Employe '(7655)))
= Vendeur
```

(DRIVER-OBJECT-CLASS <object>) **[SUBR à 1 argument]**

Renvoie la classe (persistante) de l'objet <object>. Si <object> est encore un défaut d'objet, consulte alors la classe de l'objet relationnel associé dans la base de données (fait appel à DRIVER-OBJECT-CLASS-IN-DATABASE pour cela).

Exemple:

```
? (DRIVER-OBJECT-CLASS (DRIVER-GET-OBJECT 'Employe '(7655)))
= Vendeur
```

(DRIVER-OBJECT-MAIN-CLASS <object>) **[SUBR à 1 argument]**

Renvoie la classe principale (persistante) de l'objet <object>.

Exemple:

```
? (DRIVER-OBJECT-MAIN-CLASS (DRIVER-GET-OBJECT 'Employe '(7655)))
= Employe
```

(DRIVER-CLASS-INSTANCE-P-IN-DATABASE <class-name> <object>)
(DRIVER-CLASS-INSTANCE-P-IN-DATABASE0 <class> <object>)
[SUBR à 2 arguments]

Teste si l'objet <object> est au moins instance de la classe de nom <class-name> (resp. l'objet <class>) dans la base de données.

Exemple:

```
? (DRIVER-GET-OBJECT 'Employe '(7655))
= driver-object-default-19
? (DRIVER-CLASS-INSTANCE-P-IN-DATABASE 'Vendeur
?               (DRIVER-GET-OBJECT 'Employe '(7655)))
= t
```

2.5 Écriture d'objets dans la base de données

Une transaction concerne un objet au moins, l'ensemble des objets de la couche virtuelle au plus. Elle débute suite à la connexion à la base ou après la validation de la transaction précédente.

(DRIVER-COMMIT-OBJECTS <objects> . <deep>)
[SUBR à 1 ou 2 arguments]

Valide la transaction courante. Si <objects> vaut t, tous les objets persistants de la couche virtuelle sont validés dans la base. Si <objects> est une liste d'objets, la validation ne s'applique à eux.

Quand une liste d'objets est précisée, deux nouvelles possibilités apparaissent, selon la valeur de l'argument *deep* :

- Si *deep* n'est pas précisé ou s'il vaut t (valeur par défaut) sont non seulement sauvés dans la base les objets indiqués, mais également l'ensemble des objets qu'ils référencent directement ou indirectement par ses champs persistants.
- Si *deep* vaut () ne sont sauvés que les objets précisés.

  Cependant, dans ce second cas, sont également écrits dans la base les objets non encore persistants directement référencés par l'un au moins des objets à valider. Si l'un d'entre eux référence lui-même un nouvel objet non persistant, ce dernier est également écrit dans la base, et ainsi de suite.

Quand une liste d'objets est précisée, elle peut comprendre des objets non encore persistants. L'application de DRIVER-COMMIT-OBJECTS sur ces objets les rend alors persistants.

La variable système DRIVER-WRITING-HELP oriente le traitement des erreurs de validation de transaction. Voir sa description ci-après.

La variable DRIVER-MESSAGE-P contrôle l'affichage de messages. Si elle vaut t (valeur par défaut), des messages sont affichés lorsque la base de données est modifiée.

Exemple:

```
? (DRIVER-COMMIT-OBJECTS (LIST (DRIVER-GET-OBJECT 'Employe '(7329))))
Mise-a-jour de l'objet smith
= (smith)
```

DRIVER-FIX-KEY-GENERATION-P **[Variable]**

Quand un objet doit être inséré dans la base (objet nouvellement persistant), DRIVER doit lui affecter des valeurs de clé pour toutes les tables élémentaires de sa classe. Quand un attribut de clé est de type numérique et si la variable système DRIVER-FIX-KEY-GENERATION-P vaut t (valeur par défaut), DRIVER interroge la base pour connaître la valeur maximale de cet attribut et donne la valeur $max + 1$ à l'attribut pour l'objet. Dans le cas contraire, DRIVER demande une valeur à l'utilisateur.

DRIVER-WRITING-HELP **[Variable]**

La variable système DRIVER-WRITING-HELP oriente le traitement des erreurs de validation de transaction. Si elle vaut t (valeur par défaut), les erreurs sont analysées et des corrections sont proposées à l'utilisateur. Si elle vaut (), l'erreur est simplement déclenchée.

(DRIVER-ROLLBACK-OBJECTS <objects>) **[SUBR à 1 argument]**

Annule toute modification effectuée pendant la transaction sur les objets précisés. Si <objects> vaut t, l'annulation porte sur l'ensemble des objets persistants de la couche virtuelle. L'annulation est obtenue en retransformant en défauts d'objet tous les objets persistants concernés.

Exemple:

```
? (DRIVER-ROLLBACK-OBJECTS (LIST (DRIVER-GET-OBJECT 'Employe '(7329))))
= (smith)
```

2.6 Persistance des schémas de correspondances

Les schémas de correspondances sont persistants grâce à l'existence d'un méta schéma qui détermine la correspondance relationnels des classes système de DRIVER.

(DRIVER-METAMAPPING) **[SUBR sans argument]**

Renvoie le méta-schéma de correspodances.

Exemple:

```
? (DRIVER-METAMAPPING)
= driver-metamapping
```

(DRIVER-LOAD-MAPPING <mapping-name> . <overwritep>) **[SUBR à 1 ou 2 arguments]**

Charge dans l'environnement le schéma de correspondances de nom <mapping-name>. Ce schéma doit être contenu dans les tables système DRIVER définies dans la base de données relationnelle.

En cas de tentative de rechargement d'un schéma, l'erreur "schéma déjà existant" est provoquée, sauf si <overwrite> est précisé à t. Dans ce cas, l'ancienne définition est perdue, écrasée par la nouvelle, ainsi que toutes les définitions qui y étaient rattachées (sauf son éventuel curseur). Toutes les structures qui le contenaient pointent maintenant sur la nouvelle.

Exemple:

```
? (DRIVER-LOAD-MAPPING 'map2)
= map2
```

(DRIVER-SAVE-MAPPING <mapping-name>)
(DRIVER-SAVE-MAPPING0 <mapping>) **[SUBR à 1 argument]**

Écrit le schéma de correspondances de nom <mapping-name> dans la base de données. L'écriture a lieu dans les tables système DRIVER définies dans la base de données relationnelle.

Exemple:

```
? (DRIVER-SAVE-MAPPING 'map1)
= map1
```

2.7 Interaction avec l'utilisateur

DRIVER-MESSAGES-P **[Variable]**

Booléen. Quand cette variable vaut t (valeur par défaut), DRIVER est autorisé à afficher des messages pour signaler toute action importante qu'il entreprend (modification d'un objet, mise-à-jour ou insertion de données relationnelles dans la base, etc...).

(DRIVER-PRINT <message>) **[SUBR à 1 argument]**

Cette fonction est utilisée par DRIVER pour afficher un message <message> destiné à l'utilisateur.

Elle peut être redéfinie.

(DRIVER-WARNING <message>) **[SUBR à 1 argument]**

Cette fonction est utilisée par DRIVER pour prévenir l'utilisateur d'un problème. La teneur de l'avertissement est précisée dans le message <message>.

Cette fonction peut être redéfinie.

(DRIVER-ASK-VALUE <message> <lvalues> <exit>)
[SUBR à 3 arguments]

Cette fonction est utilisée par DRIVER pour demander à l'utilisateur de lui fournir une valeur. La demande est exprimée dans le message <message>. Éventuellement, une liste de valeurs possibles <lvalues> peut être précisée; si toute valeur est autorisée, <lvalues> vaut (). Quand DRIVER autorise l'annulation de la demande, il fournit par <exit> un nom d'échappement à utiliser pour ce cas.

Cette fonction peut être redéfinie.

2.8 Divers

(DRIVER-ALL-LINKED-OBJECTS <objects>) [SUBR à 1 argument]

Renvoie la liste des objets référencés directement ou indirectement par l'ensemble d'objets <objects>.

Exemple:

```
? (DRIVER-GET-OBJECT 'Employe '(7654))
= martin
? (DRIVER-ALL-LINKED-OBJECTS (LIST (DRIVER-GET-OBJECT 'Employe '(7654))))
= (martin blake driver-object-default-10 sales driver-object-default-8
driver-object-default-3 driver-object-default-9 allen driver-object-default-5
driver-object-default-6 driver-object-default-17 driver-object-default-19
driver-object-default-20 driver-object-default-22 driver-object-default-23
driver-object-default-24 driver-object-default-25)
```

DRIVER-COMPACT-KEY-SEPARATOR [Variable]

Contient le code du caractère utilisé comme séparateur dans les clés compactes en interne à DRIVER. Par défaut, DRIVER-COMPACT-KEY-SEPARATOR vaut #// (code du caractère " / ").

La macro DRIVER-DEFCLASSMAP génère de nouveaux noms de tables logiques pour la correspondance de chaque classe afin d'obtenir des espaces de nom indépendants.

Lors de la description des correspondances de la classe `nom-class`, DRIVER-DEFCLASSMAP substitue à la table `nom-table` une nouvelle table de nom :

```
(CATENATE (DRIVER-CREATE-SYNONYM nom-class) '@ nom-table)
```

(DRIVER-CREATE-SYNONYM <symbol>) **[SUBR à 1 argument]**

Crée et renvoie un synonyme pour le symbole <symbol>

Exemple :

```
? (DRIVER-CREATE-SYNONYM 'toto)
= t119
```

(DRIVER-FIND-SYNONYM <symbol>) **[SUBR à 1 argument]**

Retrouve le synonyme défini pour le symbole <symbol>

Exemple :

```
? (DRIVER-FIND-SYNONYM 'toto)
= t119
```

(DRIVER-FIND-SYNONYM-KEY <synonym>) **[SUBR à 1 argument]**

Retrouve le symbole pour lequel a été généré le synonyme <synonym>.

Exemple :

```
? (DRIVER-FIND-SYNONYM-KEY 't119)
= toto
```

3 Filtrage des champs en lecture et écriture

Des filtres peuvent être définis pour modifier des données issues de la base avant leur affectation dans le champ d'un objet. De manière similaire, le contenu d'un champ peut être modifié avant d'être reporté dans la base de données. Ces filtres sont mis en place sous forme de méthodes qui sont invoquées lors des transferts de données.

Les méthodes **driverloading** sont invoquées pour filtrer une donnée issue de la base et destinée à être placée dans le champ d'un objet. Inversement, les méthodes **driverwriting** sont invoquées pour filtrer un contenu de champ à reporter dans la base de données.

Le filtre est défini lors de la création du champ avec l'une des fonctions DRIVER-DEFCLASS ou DRIVER-CREATE-FIELD en utilisant l'option *filter* dont la syntaxe est :

(filter <filter-name>)
ou
(filter (<filter-name> [class] [field] [object]))

Cette déclaration implique la définition par l'utilisateur des deux méthodes #:driver loading:<filter-name> et #:driverwriting:<filter-name> qui assureront le filtrage des valeurs entre le champ correspondant et la base de données.

Si le mot-clé filter introduit le seul symbole <filter-name>, les méthodes seront invoquées avec comme unique argument la valeur à filtrer. Si au contraire il introduit une liste de paramètres, le premier doit être le nom du filtre <filter-name> suivi d'un ou plusieurs mots réservés parmi *class*, *field* et *object*. Dans ce cas, les méthodes seront invoquées avec comme premier argument la valeur à filtrer, puis, selon les mots réservés choisis, l'objet DRIVER classe de l'instance à laquelle appartient le champ, l'objet DRIVER field correspondant au champ concerné, ou encore l'instance concernée elle-même. Le passage de ces différents objets avec la valeur à filtrer permet de tenir compte du contenu de certains de leurs champs pour déterminer la valeur renvoyée par le filtre.

La fonction DRIVER-FIELD-LOADING-ORDER permet d'établir l'ordre de chargement des champs d'un objet. Un intérêt de pouvoir choisir cet ordre peut être d'utiliser le contenu de certains champs de l'objet en cours de chargement – champs à charger en priorité donc – pour en construire d'autres. Un des exemples illustre cette possibilité.

Un certain nombre de types de champ atomiques ont des filtres qui leur sont associés par défaut. Ces types sont *symbol*, *float*, *fix*, *integer* et *string*. Il n'est donc pas nécessaire de redéfinir un filtre par exemple pour un champ atomique de type symbole associé à un attribut relationnel de type string. Bien entendu, l'utilisateur est libre de le faire s'il le souhaite.

Exemple:

```
? (DRIVER-ADD-FIELD-TYPE 'date)
= date
```

```
? (DRIVER-CREATE-FIELD 'Employe 'date_embauche 'atom 'date)
= date_embauche
? (DRIVER-MAP-FIELD 'Employe 'date_embauche '(emp bdate)
?                   () '(filter date1))
= t
? (de #:driverloading:date1 (datestr)
?     (build-date-from-string datestr)) ; user function call
= #:driverloading:date1
? (de #:driverwriting:date1 (date)
?     (build-date-string date)) ; user function call
= #:driverwriting:date1
? (DRIVER-ADD-FIELD-TYPE 'misc-type)
= misc-type
? (DRIVER-CREATE-FIELD 'Adresse 'datatype 'atom 'symbol)
= datatype
? (DRIVER-CREATE-FIELD 'Adresse 'data 'atom 'misc-type)
= data
? (DRIVER-MAP-FIELD 'Adresse 'datatype '(address datatype) ())
= t
? (DRIVER-MAP-FIELD 'Adresse 'data '(address userdata)
?                   '(filter (datatyping object)))
= t
? (de #:driverloading:datatyping (data obj)
?     (selectq (send 'datatype obj)
?              (symbol (symbol () data))
?              (float (convert-string-to-float data))
?              (t data)))
= #:driverloading:datatyping
? (de #:driverwriting:datatyping (data obj)
?     (string data))
= #:driverwriting:datatyping
```

DRIVER-SYSTEM-FIELD-FILTERS [Variable]

Contient la liste des filtres système reconnus par DRIVER. Ces filtres particuliers ne sont pas gérés à l'aide de méthodes driverloading et driverwriting.

Exemple :

```
? DRIVER-SYSTEM-FIELD-FILTERS
= (() integer fix float string symbol)
```

(DRIVER-FILTERS) [SUBR sans argument]

Renvoie la liste des déclarations de filtres effectuées dans le schema courant. Cette

liste comprend les filtres système utilisés.

Exemple :

```
? (DRIVER-FILTERS)
= (symbol float fix string integer)
```

(DRIVER-FIND-FILTER-FILTERS <filter-name>) **[SUBR à 1 argument]**

Retourne la liste des champs utilisant le filtre de nom <filter-name>.

Exemple :

```
? (DRIVER-FIND-FILTER-FIELDS 'symbol)
= (nom prenom num-ss qualif tel-prive situ-famille nom sites nom
  ville etat plaque modele nom genre nom type)
```

(DRIVER-VALID-FIELD-FILTER <filter-decl>) **[SUBR à 1 argument]**

Vérifie que la déclaration de filtre <filter-decl> est correcte. Les deux méthodes `driverloading` et `driverwriting` doivent avoir été créées. En cas d'incorrection, une erreur est déclenchée.

Exemple :

```
? (DRIVER-VALID-FIELD-FILTER 'symbol)
= symbol
? (DRIVER-VALID-FIELD-FILTER '(symbol object))
** driver-valid-field-filter : Filter function arguments and filter
 declaration mismatch : (symbol object)
```

(DRIVER-FIELD-FILTER-VALIDATION <filter-name>)
[SUBR à 1 argument]

Vérifie toutes les déclarations d'utilisation du filtre de nom <filter-name>. En cas d'incorrection, une erreur est déclenchée.

Exemple :

```
? (DRIVER-FIELD-FILTER-VALIDATION 'symbol)
= symbol
```

(DRIVER-LOADING-TYPE-FILTER <field> <object> <value>)
[SUBR à 3 arguments]

Cette fonction fait passer la valeur <value> dans le filtre en lecture éventuellement défini pour le champ <field>. La valeur qu'elle retourne est ensuite affectée dans le champ de l'objet <object> en cours de chargement.

C'est elle qui gère l'application des filtres système ou des méthodes driverloading. Il est possible mais déconseillé de la redéfinir.

(DRIVER-SAVING-TYPE-FILTER <attr> <field> <object> <value>)
[SUBR à 4 arguments]

Cette fonction fait passer la valeur <value> dans le filtre en écriture éventuellement défini pour le champ <field>. La valeur qu'elle retourne est ensuite traitée pour écriture dans la base de données. Quand le champ a pour correspondance un attribut, <attr> est cet attribut, sinon elle vaut (). <object> est l'objet en cours d'écriture.

C'est cette fonction qui gère l'application des filtres système ou des méthodes driverwriting. Il est possible mais déconseillé de la redéfinir.

4 Utilisation d'un nouveau modèle objet

DRIVER peut être utilisé avec de nombreux modèles objets. La description du modèle choisi s'effectue en définissant un certain nombre de méthodes qui permettront au système de créer une classe, une instance, d'affecter un champ, etc.

4.1 Manipulation des modèles objet

Un modèle objet {USER-OBJECT-MODEL} à utiliser doit être déclaré sous forme d'une classe MicroCeyx, sous-classe de Driver-Object-model, classe racine des modèles objet.

Exemple avec SMECI :

```
(DEFTCLASS {Driver-Object-model}:Smeci-object-model)
(DE {Smeci-object-model}:prin (o) (prin "Smeci-object-model"))
```

DRIVER-ROOT-OBJECT-MODEL **[Variable]**

Cette variable contient un objet instance de la classe Driver-Object-model, classe racine des modèles objet. Elle ne doit pas être modifiée, sous peine de forte perturbation du système.

DRIVER-MICROCEYX-OBJECT-MODEL **[Variable]**

Cette variable contient un objet instance de la classe MicroCeyx, sous-classe de Driver-Object-model. Elle ne doit pas être modifiée, sous peine de forte perturbation du système.

DRIVER-USER-OBJECT-MODEL **[Variable]**

Cette variable doit contenir le modèle objet client courant. Elle est consultée à chaque création de schéma de correspondances pour déterminer de quel modèle seront les futurs objets à charger de la base correspondante. Par défaut, DRIVER-USER-OBJECT-MODEL contient un objet instance de la classe *MicroCeyx*. MicroCeyx est donc le modèle objet utilisé par défaut.

Exemple avec SMECI :

```
(setq DRIVER-USER-OBJECT-MODEL (OMAKEQ {Smeci-object-model}))
```

(DRIVER-CURRENT-OBJECT-MODEL) **[SUBR sans argument]**

Renvoie le modèle objet courant. Ce modèle courant est le modèle du schéma courant, s'il y en a un, ou le modèle référencé par DRIVER-USER-OBJECT-MODEL

dans le cas contraire. C'est le modèle utilisé par DRIVER pour toutes ses opérations effectuées sur les objets client.

Exemple avec SMECI :

```
? (DRIVER-CURRENT-OBJECT-MODEL)
= Smeci-object-model
```

4.2 L'interface fonctionnelle objet

Les exemples de définition de méthode sont présentées pour une utilisation de DRIVER avec SMECI. La définition de l'interface fonctionnelle pour Microceyx est donnée en annexe.

({USER-OBJECT-MODEL}:DEFINE-CLASS <o-model> <class>) **[SUBR à 2 arguments]**

Cette fonction doit créer la classe correspondant à la classe DRIVER <class> dans le modèle objet USER-OBJECT-MODEL.

Exemple avec SMECI (La méthode complète, notamment les fonctions :smeci-field-def et :class-fields-to-define sont présentées en annexe §8) :

```
(de {Smeci-object-model}:define-class (object-model class)
    (tag bad-allocation
         (:define-class %class))
    t)

(de :define-class (class)
    (if (S-category-p (:class-name class))
        (S-kill-category (:class-name class)))
    (apply 'defScategory
           (mcons (:class-name class)
                  (or (if (send 'super class)
                          (send 'name (send 'super class)))
                      'S.Object)
                  (mapcar ':smeci-field-def
                          (:class-fields-to-define class)
                          (cirlist class))))
    t)

(de :class-name (class)
    (if (eq (send 'name class) 'driver-object-default)
        'S.Object-default
        (send 'name class)))
```

({USER-OBJECT-MODEL}:EXISTING-CLASS-P <o-model> <class>)
[SUBR à 2 arguments]

Méthode devant déterminer si la classe correspondant à la classe DRIVER <class> existe dans le modèle objet USER-OBJECT-MODEL.

Exemple avec SMECI :

```
(de {Smeci-object-model}:existing-class-p (object-model class)
    (S-category-p (:class-name class)))
```

({USER-OBJECT-MODEL}:OMAKE <o-model> <class>)
[SUBR à 2 arguments]

Méthode devant créer et renvoyer une nouvelle instance de la classe correspondant à la classe DRIVER <class> dans le modèle objet USER-OBJECT-MODEL.

Exemple avec SMECI :

```
(de {Smeci-object-model}:omake (object-model class)
    (S-create-object (:class-name class)))
```

({USER-OBJECT-MODEL}:READ-FIELD-VALUE <o-model>
<message> <object>)
[SUBR à 3 arguments]

Fonction devant retourner le contenu du champ <field-name> de l'objet <object>.

Exemple avec SMECI :

```
(de {Smeci-object-model}:read-field-value (object-model message object)
    (S-get-value message object))
```

({USER-OBJECT-MODEL}:SET-FIELD-VALUE <o-model> <message>
<object> <value>)
[SUBR à 4 arguments]

Fonction devant affecter le contenu du champ <field-name> de l'objet <object> avec la valeur <value>.

Exemple avec SMECI :

```
(de {Smeci-object-model}:set-field-value
```

```
                              (object-model message object value)
    (ifn (and (memq message '(name nom))
              (eq ({Smeci-object-model}:read-field-value
                                    object-model message object)
                  value))
         (S-set-value message object value))
    value)
```

({USER-OBJECT-MODEL}:SEND-MESSAGE <o-model> <message> <object> <args>)
[SUBR à 4 arguments]

Méthode devant envoyer le message <message> à l'objet <object> avec les arguments <args> dans le modèle objet USER-OBJECT-MODEL.

Exemple avec SMECI :

```
(de {Smeci-object-model}:send-message
                             (object-model message object args)
    (apply 'send (mcons message object args)))
```

({USER-OBJECT-MODEL}:CLASS-INSTANCE-P <o-model> <object> <class>)
[SUBR à 3 arguments]

Prédicat devant déterminer si l'objet <object> est instance de la classe correspondant à la classe DRIVER <class> dans le modèle objet USER-OBJECT-MODEL.

Exemple avec SMECI :

```
(de {Smeci-object-model}:class-instance-p (object-model object class)
    (S-isa-p object (:class-name class)))
```

({USER-OBJECT-MODEL}:CHANGE-OBJECT-CLASS <o-model> <newclass> <object>)
[SUBR à 3 arguments]

Fonction devant faire transiter l'objet <object> de son ancienne classe vers la classe correspondant à la classe DRIVER <newclass> dans le modèle objet USER-OBJECT-MODEL.

Exemple avec SMECI :

```
(de {Smeci-object-model}:change-object-class
                               (object-model newclass object)
    (if (or (driver-object-default-p object)
            (eq newclass (driver-object-default-class)))
        (S-change-category object 'S.Object))
    (S-change-category object (send 'name newclass)))
```

({USER-OBJECT-MODEL}:ALL-CLASS-OBJECTS <o-model> <class>)
[SUBR à 2 arguments]

Fonction devant renvoyer l'ensemble des instances de la classe correspondant à la classe DRIVER <class> dans le modèle objet USER-OBJECT-MODEL.

Exemple avec SMECI :

```
(de {Smeci-object-model}:all-class-objects (class)
    (S-all-objects (:class-name class)))
```

5 Exemple de base relationnelle, smecidemo

Cette base relationnelle est celle utilisée dans l'exemple de session d'utilisation présenté ci-après. Elle a été définie sous INGRES Version 6.3/02 sur sun3 et sun4.

Sont présentées d'abord les structures des relations. Suivent ensuite leurs données, l'ensemble étant disposé sous forme de tableaux.

emp		Table des employés
empno	entier	Numéro de l'employé, clé de la relation emp
ename	20 caractères	Nom de l'employé
fname	20 caractères	Prénom
job	20 caractères	Emploi dans l'entreprise
mgr	entier	Numéro du chef de l'employé
sal	réel	Salaire
com	réel	Commission, pour certains employés
deptn	entier	Numéro du département de l'employé
responsible_for	20 caractères	référence d'un objet

person		Table des personnes
pname	20 caractères	Nom de la personne
fname	20 caractères	Prénom, nom et prénom forme la clé de la relation
ssnum	15 caractères	Numéro de sécurité sociale
position	10 caractères	Situation de famille
husband_wife	10 caractères	Prénom du conjoint (si marié)
phone	12 caractères	Numéro de téléphone
car	entier	Numéro de la voiture de la personne

dept		Table des départements
deptno	entier	Numéro de département, clé de la relation
dname	20 caractères	Nom du département

deptsite		Tables des sites des départements
deptno	entier	Numéro de département, clé de la relation
sname	20 caractères	Localisation

salgrade		Table donnant le grade en fonction du salaire
grade	entier	grade (nombre), clé de la relation
losal	entier	salaire inférieur
hisal	entier	salaire supérieur
project		Table des projets de recherche
projno	entier	Numéro de projet, clé de la relation
pname	20 caractères	Nom du projet
budget	entier	Son budget

emproj		Table gérant la relation n-m entre employés et projets, un projet étant constitués d'un certain nombre d'employés, et un employé pouvant appartenir à plusieurs projets
projno	entier	Numéro du projet
empno	entier	Numéro de l'employé
		projno et empno forment la clé de la relation
address		Table des adresses
empno	entier	Numéro de l'employé de l'adresse en question, clé de la relation
num	entier	Numéro dans la rue
street	20 caractères	Nom de la rue
zip	entier	Code postal
city	12 caractères	Nom de la ville
state	5 caractères	Nom de l'état
vehicle		Table des véhicules
vnum	entier	Numéro de véhicule, clé de la relation
model	15 caractères	Modèle du véhicule
licence	15 caractères	Immatriculation. valeurs uniques
year	entier	Année de construction
defproducts		Tables des produits gérés par les départements
deptno	entier	Numéro de département
pref	20 caractères	Référence d'objet produit
software		Tables des logiciels développé par la société
softnum	entier	Numéro de logiciel
sname	20 caractères	Nom du logiciel
category	20 caractères	Type de logiciel
year1	entier	Date de début de développement
s_year	entier	Date de commercialisation
hardware		Table du matériel développé par la société
hardnum	entier	Numéro de matériel
hname	20 caractères	Nom du produit
type	20 caractères	Type de matériel

Table **emp**

empno	ename	fname	job	mgr	sal	com	deptn	responsible_for
7839	king	paul	president		5000.		10	project/102
7566	jones	eric	manager	7839	2975.		20	dept/20
7698	blake	harold	manager	7839	2850.		30	dept/30
7782	clark	john	manager	7839	2450.		10	dept/10
7902	ford	john	analyst	7566	3000.		20	project/101
7788	scott	pit	analyst	7566	3000.		20	project/103
7499	allen	jack	salesman	7698	1600.	300.	30	software/743
7521	ward	peter	salesman	7698	1250.	1400.	30	hardware/53
7654	martin	georges	salesman	7698	1250.	1400.	30	
7655	james	peter	salesman	7698	1350.	1000.	30	
7934	miller	paul	clerk	7782	1300.		10	
7329	smith	john	clerk	7902	800.		20	

Table **person**

pname	fname	ssnum	position	husband_wife	phone	car
king	paul	A34F4	married	anita	40-223233	1232
jones	eric	B7C123	married		40-783539	1975
scott	helen	NJGOFBE	married	pit	44-003231	1928
king	anita	NVJS5	married	paul	40-223233	
blake	harold	A473SE	celibate		44-183211	1429
clark	john	462SQ1	married	laura	40-893723	1230
clark	laura	G35H89	married	john	40-893756	
miller	rita	9043TU	married	paul	43-127772	
ford	laura	J9845G2	married	john	40-374845	4328
ford	john	6D3210	married	laura	40-374845	4328
scott	pit	A435C	married	helen	44-003231	1928
allen	jack	76373Y	celibate		78-383726	2004
ward	peter	128347	married	susie	40-378467	1253
ward	suzie	OIRGNE	married	peter	40-378502	
james	peter	K4G262	separate		40-449323	
martin	georges	234FS1	celibate		40-276951	3005
miller	paul	95Z0H5	married	rita	43-127772	1276
smith	john	J3847Y2	celibate		44-332047	1550

Table **dept**	
deptno	dname
10	accounting
20	research
30	sales

Table **deptsite**	
deptno	sname
10	New-York
20	Boston
30	Chicago
10	Chicago
30	Los-Angeles
30	Austin

Table **deptproducts**	
deptno	pref
20	software/743
30	software/874
30	software/743
30	hardware/53

Table **salgrade**		
grade	losal	hisal
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

Table **project**		
projno	pname	budget
102	beta	175000
103	gamma	95000
101	alpha	250000

Table **emproj**	
projno	empno
101	7566
101	7788
101	7902
101	7329
101	7839
102	7698
102	7521
102	7654
102	7499
102	7566
103	7782
103	7934
103	7566

Table **software**				
softnum	sname	category	year1	s_year
743	Pay1	pay	1974	1975
874	Accountancy1	book-keeping	1975	1977

Table **hardware**		
hardnum	hname	type
53	hbd123	computer

Table **address**					
empno	num	street	zip	city	state
7839	345	5th avenue		New-York	NY
7566	34	Maple street		Boston	MA
7698	12	Vermont street		Chicago	
7782	125	86th street		New-York	NY
7902					
7788					
7521					
7654					
7655					
7934	24	56th street		Rego Park	NY
7329	3	Highgate road		Boston	MA
7499					

Table **vehicle**			
vnum	model	licence	year
1232	Ford-1300	FD3840	1977
1975	Chryler-303	HG8943	1973
1429	Peugeot-205	PO2354	1989
1230	Ford-1300	LD7646	
4328	Ekton-730	HY4738	
1928	Peugeot-305	AW8593	1985
2004	BMW-730	UU4853	1990
1253	Toyota-1900	PV3824	
3005	Ford-1300	GH4500	
1276	Chrysler-303	KJ4753	
1550	Toyota-1800	XD4007	

6 Exemple de schéma de correspondances

Ceci est un exemple de schéma de correspondances établi pour l'exploitation de la base de données smecidemo.

Ce schéma est contenu dans le fichier `example.ll`.

```
; -----------------------------------------------------
; Mapping Lisp et Bases de donnees relationnelles
; example.ll - Exemple de schema de correspondances
; INRIA/CERMICS - 17 juillet 1991
; -----------------------------------------------------

;-----------------------------------------------------
; Name           : Mapping creation
;-----------------------------------------------------

(driver-create-mapping 'map1 t)

;-----------------------------------------------------
; Name           : Table definitions
;-----------------------------------------------------

(driver-deftable emp
   (empno           integer 2 keypart)
   (ename           string 20 not-null)
   (fname           string 20 ())
   (job             string 20 ())
   (mgr             integer 2 ())
   (sal             float   8 ())
   (com             float   8 ())
   (deptn           integer 2 ())
   (responsible_for string 20 ()))

(driver-deftable person
   (pname    string 20 keypart)
   (fname    string 20 keypart)
   (ssnum    string 15 unique)
   (position string 10 ())
   (phone    string 12 ())
   (car      integer 2 ()))

(driver-deftable salgrade
   (grade integer 2 keypart)
   (losal integer 2 ())
   (hisal integer 2 ()))

(driver-deftable dept
   (deptno integer 2 keypart)
   (dname  string 20 ()))
```

```
(driver-deftable deptsite
   (dept  integer 2 keypart)
   (sname string 20 keypart))

(driver-deftable deptproducts
   (deptno integer  2 keypart)
   (pref   string  20 keypart))

(driver-deftable project
   (projno integer 2 keypart)
   (pname  string 20 ())
   (budget integer 2 ()))

(driver-deftable emproj
   (projno integer 2 keypart)
   (empno  integer 2 keypart))

(driver-deftable address
   (empno  integer 2 keypart)
   (num    integer 2 ())
   (street string 20 ())
   (zip    integer 2 ())
   (city   string 12 ())
   (state  string  5 ()))

(driver-deftable vehicle
   (vnum    integer 2 keypart)
   (model   string 15 ())
   (licence string 15 unique)
   (year    integer 2 ()))

(driver-deftable software
   (softnum  integer  2 keypart)
   (sname    string  20 ())
   (category string  20 ())
   (year1    integer  2 ())
   (s_year   integer  2 ()))

(driver-deftable hardware
   (hardnum integer  2 keypart)
   (hname   string  20 ())
   (type    string  20 ()))


;-----------------------------------------------------
; Name           : Class descriptions
;-----------------------------------------------------

(driver-defclass Employe ()
   (nom       atom   symbol (localp ()))
```

```
   (prenom    atom   symbol)
   (num-ss    atom   symbol)
   (qualif    atom   symbol)
   (chef      object Employe)
   (responsable-de object2)
   (salaire   atom   float
        (constraint (lambda (s) (and (>= s 700.) (<= s 9999.)))))
   (grade     atom   fix (initval 0))
   (dpt       object Departement)
   (adresse   object Adresse)
   (tel-prive atom   symbol)
   (vehicule  object Vehicule))

(driver-defclass Vendeur Employe
   (qualif (constraint (lambda (qual) (eq qual 'salesman))))
   (situ-famille atom      symbol)
   (commission   atom      float)
   (sal-total    monotexpr float))

(driver-defclass Cadre Employe
   (qualif (constraint (lambda (q) (memq q '(manager president))))))

(driver-defclass President Cadre
   (qualif (constraint (lambda (q) (eq q 'president)))))

(driver-defclass Chef-de-service Cadre
   (qualif (constraint (lambda (q) (eq q 'manager)))))

(driver-defclass Departement ()
   (nom       atom       symbol (localp ()))
   (produits  object2set)
   (sites     atomset    symbol)
   (chefs     ordobjset  Cadre)
   (personnel objectset  Employe)
   (moy-sal   multitexpr float))

(driver-defclass Projet ()
   (nom       atom       symbol (localp ()))
   (budget    atom       float)
   (employes  ordobjset  Employe)
   (sal-total multitexpr float))

(driver-defclass Adresse ()
   (numero atom fix)
   (rue atom string)
   (ville atom symbol)
   (etat  atom symbol))

(driver-defclass Vehicule ()
   (plaque atom symbol)
```

```
   (modele atom symbol))

(driver-defclass Software ()
   (nom         atom symbol (localp ()))
   (genre       atom symbol)
   (annee1      atom integer)
   (annee-vente atom integer))

(driver-defclass Hardware ()
   (nom  atom symbol (localp ()))
   (type atom symbol))

;-----------------------------------------------------
; Name           : Map definitions
;-----------------------------------------------------

(driver-defclassmap Employe emp
   (letjoins
       ((j1 (emp (p person)
            ((lambda (a1 a2 a3 a4)
                (and (eq a1 a3) (eq a2 a4)))
             (emp ename) (emp fname) (p pname) (p fname))))
        (j2 (emp (emp1 emp)
            ((lambda (a1 a2) (eq a1 a2))
               (emp mgr) (emp1 empno))))
        (j3 (emp (sg salgrade)
            ((lambda (a1 a2 a3)
               (and (>= a1 a2) (<= a1 a3)))
             (emp sal) (sg losal) (sg hisal))))
        (j4 (emp dept
            ((lambda (a1 a2) (eq a1 a2))
             (emp deptn) (dept deptno))))
        (j5 (emp address
            ((lambda (a1 a2) (eq a1 a2))
             (emp empno) (address empno))))
        (j6 (p vehicle
            ((lambda (a1 a2) (eq a1 a2))
             (p car) (vehicle vnum)))))
       (fields (nom            (emp ename)           () (readonly t))
               (prenom         (emp fname)           () (readonly t))
               (num-ss         (p ssnum)             (j1))
               (qualif         (emp job)             ())
               (chef           ()                    (j2))
               (responsable-de (emp responsible_for) ())
               (salaire        (emp sal)             ())
               (grade          (sg grade)            (j3) (readonly t))
               (dpt            ()                    (j4))
               (adresse        ()                    (j5))
               (tel-prive      (p phone)             (j1))
               (vehicule       ()                    (j1 j6)))))
```

```
(driver-defclassmap Vendeur
   (fields (situ-famille (p position)      ((Employe emp p)))
           (commission   (emp com) ())
           (sal-total    ((lambda (a1 a2) (+ a1 a2))
                          (emp sal) (emp com)) ())))

(driver-defclassmap Departement dept
   (letjoins
       ((j1 (dept (e emp)
            ((lambda (attr1 attr2) (eq attr1 attr2))
             (dept deptno) (e deptn))))
        (j2 (dept (ds deptsite)
            ((lambda (a b) (eq a b))
             (dept deptno) (ds dept))))
        (j3 (dept (dp deptproducts)
            ((lambda (a b) (eq a b))
             (dept deptno) (dp deptno)))))
       (attrconstraint ((lambda (no) (> no 0)) (dept deptno))
                       ())
       (fields (nom       (dept dname)   ())
               (produits  (dp pref)      (j3))
               (sites     (ds sname)     (j2))
               (chefs     ()             (j1)
                  (orders ((e sal) desc)
                          ((e ename) asc))
                  (readonly t))
               (personnel ()             (j1) (readonly t))
               (moy-sal   ((lambda (a1) (avg a1)) (e sal))
                                         (j1)))))

(driver-defclassmap Projet project
   (letjoins
       ((j1 (project emp
            ((lambda (a1 a2 a3 a4)
               (and (eq a1 a2) (eq a3 a4)))
             (project projno) (emproj projno)
             (emproj empno) (emp empno)))))
       (fields (nom       (project pname)  ())
               (budget    (project budget) ())
               (employes  ()               (j1) (orders ((emp empno) asc)))
               (sal-total ((lambda (a) (sum a)) (emp sal)) (j1)))))

(driver-defclassmap Adresse address
   (fields (numero (address num)    ())
           (rue    (address street) ())
           (ville  (address city)   ())
           (etat   (address state)  ())))

(driver-defclassmap Vehicule vehicle
```

```
   (fields (plaque (vehicle licence) ())
           (modele (vehicle model)  ())))

(driver-defclassmap Software software
   (fields (nom         (software sname)    ())
           (genre       (software category) ())
           (annee1      (software year1)    ())
           (annee-vente (software s_year)   ())))

(driver-defclassmap Hardware hardware
   (fields (nom  (hardware hname) ())
           (type (hardware type)  ())))


;-----------------------------------------------------
; Name           : Mapping compilation
;-----------------------------------------------------

(driver-compile-mapping 'map1)
```

7 Exemple de session d'utilisation

Cette exemple de session d'utilisation est effectuée avec le SGBD Ingres v6. La base utilisée est `smecidemo` dont les relations et leurs contenus sont donnés en annexe (cf. §5).

Le schéma de correspondances utilisé est `map1`, présenté précédemment. Il est contenu dans le fichier `example.ll` (cf. §6).

Après chargement d'un environnement Le_lisp (lelisp, aida, smeci, ...), ...

```
; Le-Lisp (by INRIA) version 15.24 ( 2/Janv/91)   [sun]
; Systeme Complice : mer 13 mars 91 10:48:22
= (31bitfloats abbrev callext compiler complice date debug defstruct
loader mc68881 messages microceyx pathname pepe pretty setf
virbitmap virtty)
?
?
? ; On se place dans le directory ou se trouve driver.ll
? !cd /u/crazy/0/smeci/lebastar/driver
= t
? ^Ldriver
Chargement d'ASQUELL...
Chargement d'ASQUELL oK
Chargement de DRIVER...
DRIVER v1.34
Chargement de DRIVER oK
= driver.ll
?
?
?
? ; Connexion a la base smecidemo
? (driver-connect 'ingres 'smecidemo)
= #:tclass:cursor:#[ingres 0 t ()]
?
? ; Chargement et definition d'un schema de correspondances
? ^Lexample
= example.ll
?
?
? (driver-all-mappings)
= (map1)
? (driver-current-mapping)
= map1
?
? (driver-existent-key-p 'Employe '(7698))
= t
? (driver-get-object 'Employe '(7698))
```

```
= driver-object-default-1
? (driver-get-object-if-loaded 'Employe '(7698))
= ()
? (driver-object-default-p
?           (driver-get-object 'Employe '(7698)))
= driver-object-default-1
? (driver-load-object-default
?           (driver-get-object 'Employe '(7698)))
= blake
? (driver-get-object 'Employe '(7698))
= blake
? (driver-get-object-if-loaded 'Employe '(7698))
= blake
?
? (driver-load-class-objects 'Departement t)
= (accounting research sales)
? (driver-get-object 'Employe '(7499))
= driver-object-default-16
? (driver-object-class-in-database (driver-get-object 'Employe '(7499)))
= Vendeur
? (driver-read-field-value-in-database 'commission
?                (driver-get-object 'Employe '(7499)))
= 400.
? (driver-set-field-value-in-database 'commission
?                (driver-get-object 'Employe '(7499)) 500.)
Mise-a-jour de l'objet S.Object-default-16
= 500.
? (driver-get-object 'Employe '(7499))
= driver-object-default-16
? (driver-read-field-value 'salaire
?                (driver-get-object 'Employe '(7499)))
= 1600.
? (driver-get-object 'Employe '(7499))
= allen
?
?
? ; Filtrer les employes parmi tous les tuples de emp
? (driver-select-objects '((lambda (obj) t) Employe))
= (((7329)) ((7499)) ((7521)) ((7566)) ((7654)) ((7655))
 ((7698)) ((7782)) ((7788)) ((7839)) ((7902)) ((7934)))
?
? ; Filtrer les vendeurs parmi tous les tuples de emp
? (driver-select-objects '((lambda (obj) t) Vendeur))
= (((7499)) ((7521)) ((7654)) ((7655)))
```

```
?
? ; Filtrer les employes qui ont une voiture de meme modele
? ; que leur chef
? (driver-select-objects
?    '((lambda (obj)
?         (eq (send 'modele (send 'vehicule obj))
?             (send 'modele (send 'vehicule (send 'chef obj)))))
?      Employe))
= (((7782)))
?
?
?
? (end)
Que Le_lisp soit avec vous.
```

8 Exemple de prise en compte d'un modèle objet

Voici un exemple de prise en compte du modèle objet de SMECI. Le listing suivant est en fait le contenu du fichier `smecidriver.ll`.

```
; ----------------------------------------------------------------------------
; Mapping Lisp et Bases de donnees relationnelles
; smecidriver.ll - Definition des methodes objet pour utilisation de Smeci V1.5
; INRIA/CERMICS - 4 septembre 1989 - Maj 21 avril 1992
; ----------------------------------------------------------------------------

(setq #:sys-package:colon 'smecidriver)

(eval-when (load eval)
(add-feature 'smecidriver)
)

(eval-when (load eval compile)
(deftclass {Driver-Object-model}:Smeci-object-model)
)

(eval-when (load eval compile)
(defvar driver-user-object-model (omakeq {Smeci-object-model}))
(defvar :dbid-sep #/@)
)

(de {Smeci-object-model}:prin (o) (prin "Smeci-object-model"))


;-----------------------------------------------------
; Name           : {Smeci-object-model}:define-class
; Date           : 09/04/89
; Author         : Franck LEBASTARD
; Arguments      : Driver-class instance
; Result         : t
; Description    : Creates Smeci categorie and declare it in the system.
; Side effects   : Does deftclass
; Export         : Yes (method)
; Document       : No
; Modified date  : No
;-----------------------------------------------------

(de {Smeci-object-model}:define-class (object-model class)
    (tag bad-allocation
         (:define-class class))
    t)

(de :define-class (class)
    (if (S-category-p (:class-name class))
        (S-kill-category (:class-name class)))
```

```
    (apply 'defScategory
           (mcons (:class-name class)
                  (or (if (ogytq Driver-class super class)
                          (:class-name (ogytq Driver-class super class)))
                      'S.Object)
                  (mapcar ':smeci-field-def
                          (:class-fields-to-define class)
                          (cirlist class)))))

(de :class-name (class)
    (if (eq (ogytq Driver-class name class) 'driver-object-default)
        'S.Object-default
        (ogytq Driver-class name class)))

(de :class-fields-to-define (class)
    (let ((fields (mapcan (lambda (field)
                            (and (ogytq Driver-field localp field)
                                 (list field)))
                          (ogytq Driver-class fields class))))
         (mapc (lambda (field)
                  (if (and (driver-field-p field)
                           (not (memq field fields)))
                      (setq fields (nconc1 fields field))))
               (mapcar 'car (ogytq Driver-class restrictions class)))
         fields))

(de :redefine-category-tree (main-class)
    (mapc '{Smeci-object-model}:define-class
          (cirlist ())
          (cons main-class (driver-class-subclasses0 main-class))))

;-----------------------------------------------------
; Name           : :smeci-field-def
; Date           : 09/04/89
; Author         : Franck LEBASTARD
; Arguments      : Driver-field object, Driver-class object
; Result         : Symbolic smeci field definition for creer-categorie funct.
; Description    : Calculates every field slot
; Side effects   : No
; Export         : No
; Document       : No
; Modified date  : No
;-----------------------------------------------------

; Les facettes inverse import unite type-lien extension ne sont pas remplies.
(de :smeci-field-def (field class)
    (mapcan (lambda (facette)
               (tag smeci-field-def
                    (list facette (send facette field class))))
            '(nom-ch allocation type domain range force-test?)))
```

```
;-----------------------------------------------------
; Name           : nom-ch defini type domain range
; Date           : 09/04/89
; Author         : Franck LEBASTARD
; Arguments      : Driver-field object, Driver-class object
; Result         : slot value
; Description    : Calculate field slots
; Side effects   : No
; Export         : Yes (methods)
; Document       : No
; Modified date  : No
;-----------------------------------------------------

(de {Driver-field}:nom-ch (field class)
    (exit smeci-field-def (list (ogytq Driver-field name field))))

; Warning : the p type can't be deducted with the only defined informations
(de {Driver-field}:allocation (field class)
    (ifn (memq field (ogytq Driver-class fields class))
         (exit smeci-field-def ())
         (if (:any-constraint-on-field class field)
             'constrained 'instance)))

(de :any-constraint-on-field (class field)
    (car (any (lambda (class1)
                 (driver-find-field-constraints-on-class0 class1 field))
              (cons class (driver-class-subclasses0 class)))))

(de {Object-field}:allocation (field class) 'instance)
(de {Object-set}:allocation (field class) 'instance)

(de {Atom-field}:type (field class)
    (ifn (memq field (ogytq Driver-class fields class))
         (exit smeci-field-def ())
         (let ((rest (:any-constraint-on-field class field)))
              (:smeci-type (ogytq Driver-field ftype field)
                           (and rest
                                (send 'domain-type
                                      (ogytq Driver-join0 operator rest)))))))

(de {Atom-set}:type (field class)
    (ifn (memq field (ogytq Driver-class fields class))
         (exit smeci-field-def ())
         (:smeci-type (ogytq Driver-field ftype field) 'set)))

(de {Object-field}:type (field class)
    (:smeci-type 'object ()))

(de {Object-set}:type (field class)
```

```
    (:smeci-type 'object 'set))

(de {Computed-field}:type (field class)
    (ifn (memq field (ogytq Driver-class fields class))
         (exit smeci-field-def ())
         (:smeci-type (ogytq Driver-field ftype field) ())))

(de :smeci-type (type1 type2)
    (selectq type1
             (string
               (selectq type2
                        (set 'l*)
                        (t '*)))
             (symbol
               (selectq type2
                        (enum 'enum-symbol)
                        (bounded 'bounded-symbol)
                        (set 'lsymbol)
                        (t 'symbol)))
             ((fix integer)
               (selectq type2
                        (enum 'enum-fix)
                        (bounded 'bounded-fix)
                        (set 'lfix)
                        (t 'fix)))
             (float
               (selectq type2
                        (enum 'enum-float)
                        (bounded 'bounded-float)
                        (set 'lfloat)
                        (t 'float)))
             (object
               (selectq type2
                        (set 'lobject)
                        (t 'object)))
             (()
               (selectq type2
                        (set 'l*)
                        (t '*)))
             (t (print "Not a good type : " type1) ())))

(de {Driver-field}:domain (field class) (exit smeci-field-def ()))
(de {Object-field}:domain (field class)
    (or (nlistp (:class-name (ogytq Object-field class field)))
        (car (:class-name (ogytq Object-field class field)))))

(de {Object-set}:domain (field class)
    (or (nlistp (:class-name (ogytq Object-set setobjclass field)))
        (car (class-name (ogytq Object-set setobjclass field)))))
```

```
(de {Driver-field}:range (field class)
    (let ((rest (cassq field (ogytq Driver-class restrictions class))))
         (ifn rest (exit smeci-field-def ())
              (or (and (ogytq Driver-class super class)
                       (eq (S-get-facet-value
                                    (:class-name
                                           (ogytq Driver-class super class))
                                    (ogytq Driver-field name field)
                                    'allocation)
                           'instance)
                       (exit bad-allocation
                             (:redefine-category-tree
                                    (driver-find-main-class0 class))))
                  (:build-smeci-range
                            (send 'smeci-range
                                  (ogytq Driver-join0 operator rest)
                                  field))))))

(de :build-smeci-range (lranges)
    (selectq (caar lranges)
             (min (list (cdar lranges) (cassq 'max lranges)))
             (max (list (cassq 'min lranges) (cdar lranges)))
             (lval (cdar lranges))
             (val (mapcan (lambda ((key . val))
                            (if (eq key 'val) (list val)))
                          lranges))))

; smeci ne gere pas les negations.
(de {Driver-sql-op}:domain-type (op))

(de {Driver-n-op}:domain-type (op)
    (any (lambda (op1) (send 'domain-type op1))
         (ogytq Driver-n-op args op)))

(de {Driver-val-link}:domain-type (op)
    (selectq (ogytq Driver-sql-op op op)
             (= 'enum)
             ((< <= > >=) 'bounded)))

(de {Driver-set-link}:domain-type (op)
    (if (eq (ogytq Driver-sql-op op op) 'in) 'enum))

(de {Driver-sql-op}:smeci-range (op field))

(de {Driver-or}:smeci-range (op field)
    (mapcan (lambda (r)
              (if (eq (car r) 'val) (list r)))
            (mapcan (lambda (op1)
                      (send 'smeci-range op1 field))
                    (ogytq Driver-n-op args op))))
```

```
(de {Driver-and}:smeci-range (op field)
    (mapcan (lambda (op1) (send 'smeci-range op1 field))
            (ogytq Driver-n-op args op)))

(de {Driver-val-link}:smeci-range (op field)
    (let ((val (driver-loading-type-filter field ()
                                           (ogytq Driver-2-op arg2 op))))
         (selectq (ogytq Driver-sql-op op op)
                  (= (list (cons 'val val)))
                  ((> >=) (list (cons 'min val)))
                  ((< <=) (list (cons 'max val))))))

(de {Driver-set-link}:smeci-range (op field)
    (if (eq (ogytq Driver-sql-op op op) 'in)
        (let ((lval (mapcar (lambda (val)
                              (driver-loading-type-filter field () val))
                            (ogytq Driver-2-op arg2 op))))
             (list (cons 'lval lval)))))

(de {Driver-field}:force-test? (field class)
    (exit smeci-field-def ()))

(de {Object-field}:force-test? (field class)
    (ifn (memq field (ogytq Driver-class fields class))
         (exit smeci-field-def ())))

(de {Object-set}:force-test? (field class)
    (ifn (memq field (ogytq Driver-class fields class))
         (exit smeci-field-def ())))

#|
(de {Driver-field}:inverse (field))

(de {Driver-field}:import (field))

(de {Driver-field}:unite (field))

(de {Driver-field}:type-lien (field))

(de {Driver-field}:extension (field))
|#


;-----------------------------------------------------
; Name           : {Smeci-object-model}:omake
; Date           : 09/11/89
; Author         : Franck LEBASTARD
; Arguments      : Driver-class object according to a Smeci categorie, from
;                : which a new instance will be created.
```

```
; Result         : A new class instance.
; Description    : Calls Smeci nouvel-objet function.
; Side effects   : On Smeci
; Export         : Yes (method)
; Document       : No
; Modified date  : No
;-----------------------------------------------------

(de {Smeci-object-model}:omake (object-model class)
    (S-create-object (:class-name class)))


;-----------------------------------------------------
; Name           : {Smeci-object-model}:set-field-value
; Date           : 03/90
; Author         : Franck LEBASTARD
; Arguments      :
; Result         :
; Description    :
; Side effects   :
; Export         : Yes (method)
; Document       :
; Modified date  :
;-----------------------------------------------------

(de {Smeci-object-model}:set-field-value (object-model msg obj value)
    (ifn (and (memq msg '(name nom))
              (eq ({Smeci-object-model}:read-field-value object-model
                                                         msg obj)
                  value))
        (S-set-value msg obj value))
    value)

(de {Smeci-object-model}:read-field-value (object-model msg obj)
    (S-get-value msg obj))


;-----------------------------------------------------
; Name           : {Smeci-object-model}:change-object-class
; Date           : 09/07/89
; Author         : Franck LEBASTARD
; Arguments      : Driver-class object, Any Smeci Objet object
; Result         : Objet object from Driver-class class.
; Description    : Changes the class of an object.
; Side effects   : On Smeci
; Export         : Yes (method)
; Document       : No
; Modified date  : 09/11/89
;-----------------------------------------------------
```

```
(de {Smeci-object-model}:change-object-class (object-model newclass obj)
    (if (or (driver-object-default-p obj)
            (eq newclass (driver-object-default-class)))
        (S-change-category obj 'S.Object))
    (S-change-category obj (:class-name newclass)))


;-----------------------------------------------------
; Name           : {Smeci-object-model}:existing-class-p
; Date           : 09/07/89
; Author         : Franck LEBASTARD
; Arguments      : Driver-class object
; Result         : t or ()
; Description    : Does a class exist
; Side effects   : No
; Export         : Yes (redefined)
; Document       : No
; Modified date  : No
;-----------------------------------------------------

(de {Smeci-object-model}:existing-class-p (object-model class)
    (S-category-p (:class-name class)))


;-----------------------------------------------------
; Name           : {Smeci-object-model}:class-instance-p
; Date           : 09/07/89
; Author         : Franck LEBASTARD
; Arguments      : An object, Driver-class object
; Result         : t or ()
; Description    : Is an object a class instance
; Side effects   : No
; Export         : Yes (redefined)
; Document       : No
; Modified date  : No
;-----------------------------------------------------

(de {Smeci-object-model}:class-instance-p (object-model object class)
    (S-isa-p object (:class-name class)))

;-----------------------------------------------------
; Name           : {Smeci-object-model}:all-class-objects
; Date           : 03/30/90
; Author         : Franck LEBASTARD
; Arguments      :
; Result         :
; Description    :
; Side effects   : No
; Export         : Yes
; Document       : No
```

```
; Modified date  : No
;-----------------------------------------------------

(de {Smeci-object-model}:all-class-objects (object-model class)
    (S-all-objects (:class-name class)))


;-----------------------------------------------------
; Name           : driver-dbms-connection
; Date           :
; Author         : Franck LEBASTARD
; Arguments      :
; Result         :
; Description    :
; Side effects   :
; Export         :
; Document       :
; Modified date  :
;-----------------------------------------------------

(de driver-dbms-connection (dbms-name user-name)
    (S-set-category-collector (:show-signature '-CAT dbms-name user-name))
    (S-set-object-collector (:show-signature '-OBJ dbms-name user-name)))

;-----------------------------------------------------
; Name           : driver-object-loading-begin, driver-object-loading-end
; Date           : 09/11/89
; Author         : Franck LEBASTARD
; Arguments      : Main class
; Result         : Void
; Description    : Updates Smeci Object Controle panel.
; Side effects   : No
; Export         : Yes (redefinition)
; Document       : No
; Modified date  : No
;-----------------------------------------------------

(de driver-object-loading-begin (mainclass)
    (:current-cursor-object-database))

(de driver-object-loading-end (mainclass)
;    (funcall '#:smecicontrol:update-interface)
)

(de :current-cursor-category-database ()
    (S-set-category-collector
         (:dbms-signature '-CAT
              (ogytq cursor dbms
                     (or (driver-current-cursor)
                         (error 'smeci-driver "No current cursor" ()))))))
```

```
(de :current-cursor-object-database ()
    (S-set-object-collector
         (:dbms-signature '-OBJ
              (ogytq cursor dbms
                     (or (driver-current-cursor)
                         (error 'smeci-driver "No current cursor" ()))))))

(de :dbms-signature (coltype dbms)
    (:show-signature coltype (ogytq dbms name dbms)
                     (ogytq database name (ogytq dbms database dbms))))

(de :show-signature (coltype nom-sgbd utilisateur)
    (catenate "SGBD" coltype (ascii :dbid-sep)
              nom-sgbd (ascii :dbid-sep)
              utilisateur))
```

9 Messages d'erreurs et autres de DRIVER

Codes	Messages d'erreur et autres de DRIVER
driverr11	"Pas de modèle objet courant"
	"No current object model"
driverr21	"Pas un schéma de correspondances"
	"Not a mapping"
driverr22	"Pas de schéma de correspondances courant"
	"No current mapping"
driverr23	"Schéma de correspondances existant"
	"Existent mapping"
driverr24	"Schéma de correspondances inconnu"
	"Unknown mapping"
driverr25	"Schéma de correspondances incomplet"
	"Mapping not complete"
driverr31	"nom de table existant"
	"Existent table name"
driverr32	"Définition de table inconnue"
	"Unknown table definition"
driverr33	"Pas une définition de table"
	"Not a table definition"
driverr34	"Table sans clé"
	"No table key"
driverr41	"Attribut existant"
	"Existent attribute"
driverr42	"Mauvais statut d'attribut"
	"Bad attribute status"
driverr43	"Mauvais type d'attribut"
	"Bad attribute type"
driverr44	"Type d'attribut existant"
	"Existing attribute type"
driverr45	"Définition d'attribut inconnue"
	"unknown attribute definition"
driverr46	"Pas une définition d'attribut"
	"Not an attribute definition"
driverr47	"Pas un des attribut de la table"
	"Not one of the table attributes"
driverr51	"Variable de table existante"
	"Existent table variable"

Codes	Messages d'erreur et autres de DRIVER
driverr52	"Variable de table inconnue"
	"Unknown table variable"
driverr53	"Pas une variable de table"
	"Not a table variable"
driverr54	"Pas une variable de table de base"
	"Not a basic table variable"
driverr55	"Variable de table non effacable"
	"Not an erasable table variable"
driverr61	"Variable d'attribut existante"
	"Existent attribute variable"
driverr62	"Variable d'attribut inconnue"
	"Unknown attribute variable"
driverr63	"Pas une variable d'attribut"
	"Not an attribute variable"
driverr64	"Pas une variable d'attribut effacable"
	"Not an erasable attribute variable"
driverr71	"Nom de classe existant"
	"Existent class name"
driverr72	"classe inconnue"
	"Unknown class"
driverr73	"Pas une classe"
	"Not a class"
driverr81	"Champ existant"
	"Existent field"
driverr82	"Type de champ inconnu"
	"Unknown field type"
driverr83	"Type de champ atomique mauvais"
	"Bad atomic field type"
driverr84	"Type de champ existant"
	"Existing field type"
driverr85	"Champ inconnu"
	"Unknown field"
driverr86	"Pas un champ"
	"Not a field"
driverr87	"Pas un champ atomique"
	"Not an atomic field"
driverr88	"Pas un champ de la classe"
	"Not a class field"

Codes	Messages d'erreur et autres de DRIVER
driverr89	"Pas un champ propre ou herité"
	"Not an own or herited field"
driverr90	"Pas un champ de la hiérarchie"
	"Not a hierarchy field"
driverr91	"Pas une variable d'attribut ou un champ propre ou hérité"
	"Not an attribute variable or an atomic own or herited field"
driverr101	"Contrainte de classe existante sur"
	"Existent class constraint on"
driverr102	"Contrainte inconnue"
	"Unknown constraint"
driverr103	"Expression invalide"
	"Invalid expression"
driverr111	"Jointure déjà existante"
	"Existent join"
driverr112	"Jointure inconsistante"
	"Inconsistent join"
driverr113	"Jointure inconnue"
	"Unknown join"
driverr114	"Pas une jointure"
	"Not a join"
driverr115	"Variable de jointure existante"
	"Existent join variable"
driverr116	"Variable de jointure indéfinie"
	"Undefined join variable"
driverr117	"Deux fois la même variable"
	"Two variables have same name"
driverr118	"Déjà un environnement 'fields' ou 'letjoins"'
	"Yet a 'fields' or 'letjoins' environment"
driverr119	"Noms trop proches"
	"Too close names"
driverr121	"Pas la table principale"
	"Not the main table"
driverr122	"Déjà une table principale"
	"Yet a main table"
driverr123	"Déjà une table secondaire"
	"Yet a sub-table"
driverr124	"Table principale indéfinie"
	"Undefined main table"

Codes	Messages d'erreur et autres de DRIVER
driverr125	"Table principale indéfinie dans la classe principale"
	"Undefined main table on main class"
driverr126	"Table déjà utilisée dans le schéma"
	"Table already used in mapping"
driverr127	"Attribut déjà utilisé dans le schéma"
	"Attribute already used in mapping"
driverr128	"Attribut de jointure déjà utilisé dans le schéma"
	"Join attribute already used in mapping"
driverr131	"Classe sans correspondance"
	"Class not mapped"
driverr132	"Classe avec correspondance déjà établie"
	"Already mapped class"
driverr133	"Pas une correspondance de classe"
	"Not mapped on a class"
driverr134	"Pas une classe principale"
	"Not a main class"
driverr135	"Pas de classe principale"
	"No main class"
driverr141	"Champ sans correspondance"
	"Field not mapped"
driverr142	"Champ avec correspondance déjà établie"
	"Already mapped field"
driverr143	"Aucun ordre spécifié"
	"No order specified"
driverr144	"Ordre specifié invalide"
	"Invalid order specification"
driverr145	"Ensemble de jointures incomplet"
	"Incomplete join set"
driverr146	"Trop de jointures"
	"Too many joins"
driverr147	"Table principale utilisée comme table secondaire dans une jointure"
	"Main table used as a sub-table in a join"
driverr148	"Jointure objet illégale"
	"Object join doesn't comply restrictions"
driverr149	"Champ read-only"
	"Read only field"
driverr150	"Ordre des classes invalide"
	"Invalid class order"

Codes	Messages d'erreur et autres de DRIVER
driverr161	"Pas un curseur"
	"Not a cursor"
driverr162	"Pas de curseur courant"
	"No current cursor"
driverr163	"Pas un objet dbms"
	"Not a dbms"
driverr164	"Objet dbms introuvable"
	"Can't find dbms"
driverr165	"Pas une clé d'objet valide"
	"Not a valid object key"
driverr166	"Pas une clé d'objet existante"
	"Not an existent object key"
driverr167	"Pas un défaut d'objet"
	"Not an object default"
driverr168	"Objet relationnel manquant"
	"Missing relational object"
driverr169	"Pas un objet persistant"
	"Not a persistent object"
driverr170	"Pas un champ de l'objet"
	"Not an object field"
driverr181	"Fonction de filtrage en chargement indéfinie"
	"Undefined loading filter function"
driverr182	"Fonction de filtrage en écriture indéfinie"
	"Undefined writing filter function"
driverr183	"Mauvais arguments des fonctions de filtrage en lecture et écriture"
	"Bad loading or writing filter function arguments"
driverr184	"Incompatibilité déclaration du filtre et arguments des fonctions de filtrage"
	"Filter function arguments and filter declaration mismatch"
driverr185	"Déclarations différentes d'utilisation du même filtre"
	"filter declaration mismatch"
driverr191	"Pas une lambda-expression"
	"Not a lambda expression"
driverr192	"Déclaration d'argument non autorisé"
	"Unauthorised argument declaration"
driverr193	"Trop de lambda-expression"
	"Too many lambda expressions"
driverr194	"Mauvais nombre de variables de lambda"
	"Bad lambda variable number"
driverr195	"Expression non autorisée"
	"Unauthorised expression"

Codes	Messages d'erreur et autres de DRIVER
driverr196	"Mauvais nombre ou type d'argument"
	"Bad argument types or number"
driverr197	"Fonction non autorisée"
	"Unauthorised function"
driverr198	"Variable non autorisée dans un not"
	"Variable unauthorised in a not"
driverr199	"Nil non autorisé"
	"Unexpected nil"
driverr200	"Second argument invalide"
	"Invalid 2nd argument"
driverr101	"Expression constante non autorisée"
	"Unauthorised constant expression"
driverr202	"Variable inconnue"
	"Unknown variable"
driverr203	"Pas un type numérique"
	"Not a numeric type"
driverr204	"Type incompatible"
	"Mismatch type"
driverr205	"Types incompatibles"
	"Incompatible types"
driverr206	"Comparaison entre un champ objet et un champ non objet"
	"Object and not object field(s) comparison"
driverr207	"Comparaison d'instances de classes incompatibles"
	"Incompatible class instance comparison"
driverr208	"Mauvais opérande"
	"Bad operand"
driverr209	"Mauvais comparateur d'objets"
	"Bad object comparator"
driverr210	"Type de champ non encore autorisé dans une expression de sélection d'objets"
	"Field type not supported yet in object selecting"
driverr211	"Variable indéfinie"
	"Unbounded variable"
driverr212	"Mauvais type de valeur de variable"
	"Bad variable value type"
driverr241	"Chargement de l'objet ~A"
	"Loading object ~A"
driverr242	"Écriture de l'objet ~A"
	"Inserting object ~A"
driverr243	"Mise-à-jour de l'objet ~A"
	"Updating object ~A"

Codes	Messages d'erreur et autres de DRIVER
driverr251	"Échec de requête de sélection : ~A"
	"DBMS select request error : ~A"
driverr252	"Échec de requête de lecture du max d'un attribut"
	"Maximum attribute value request error"
driverr253	"Échec de requête d'insertion : ~A"
	"DBMS insert request error : ~A"
driverr254	"Échec de requête de mise-à-jour : ~A"
	"DBMS update request error : ~A"
driverr261	"Valeur de mauvais type pour l'attribut ~A : ~A. Non sauvée."
	"Bad type value for ~A attribute : ~A. Not saved."
driverr262	"~A ne peut être sauvé comme valeur du champ ~A de l'objet ~A"
	"~A can't be saved in field ~A of ~A object"
driverr263	"Incompatibilité entre les contraintes de clé et de classe"
	"Incompatible key and class constraints"
driverr264	"Incohérence dans les contraintes d'écriture"
	"Inconsistency in writing constraints"
driverr265	"Incohérence dans les contraintes d'écriture : ~A"
	"Inconsistency in writing constraints : ~A"
driverr266	"Abandon des contraintes de clé : ~A"
	"Key constraints forsaked : ~A"
driverr267	"La valeur du champ ne vérifie pas les contraintes"
	"Field value doesn't comply constraints"
driverr268	"La valeur du champ ~A de l'objet ~A ne vérifie pas les contraintes"
	"~A Field value of object ~A doesn't comply constraints"
driverr269	"Objet ~A, champ ~A, nouvelle valeur : ~A"
	"Object ~A, field ~A, new value : ~A"
driverr270	" Annulation du commit"
	" Commit abortion "
driverr301	"Nouveau"
	"New"
driverr302	"Sélectionnez un objet"
	"Select an object"
driverr303	"Donner une valeur au champ ~A de l'objet ~A vérifiant ~A"
	"Give a value for the ~A field of object ~A, complying ~A"
driverr304	"Donnez une valeur à l'attribut ~A de type ~A, vérifiant ~A"
	"Give a value for the ~A attribute (type ~A), complying ~A"
driverr305	" Entrer une valeur "
	" Enter a value "
driverr306	" D'accord "
	" OK "

Codes	**Messages d'erreur et autres de** DRIVER
driverr401	"Pas un symbole"
	"Not a symbol"
driverr402	"Pas une liste"
	"Not a list"
driverr403	"Argument incorrect"
	"Invalid argument"
driverr404	"Option invalide"
	"Invalid option"

10 Index des fonctions et des variables

Table des matières