```
/
*****************************************************************************
******************/
/*  Student Names and IDs :
                          */
/*  ======================
                          */
/*
                          */
/*
                          */
/*
                          */
/*                ---------------------------
-----------------------          */
/*
                          */
/*
                          */
/*                ---------------------------
-----------------------          */
/*
                          */
/*
                          */
/
*                 ---------------------------      ------------------
-----             */
/*
                          */
/*
                          */
/*                ---------------------------
-----------------------          */
/*
                          */
/*
                          */
/*                ---------------------------
-----------------------          */
/*
                          */
/*
                          */
/*
                          */
/*
                          */
```

```
/*  Assignment        : #1
                        */
/*
                        */
/*  Session           : Summer 2016
                */
/*
                */
/*  Course - Section : COMP348 - AA
                */
/*
                */
/*  Language          : Prolog
                */
/*
                */
/*  File name          : C348SA1.pl
                */
/*
                */
/*  Professor         : Mohamed Taleb
                */
/*
                */
/*
                */
/*
                */
/*
                */
/*            Concordia University - Montreal
                */
/
**************************************************************************
******************/

/*****************************************************************/
/* This application deals with a relational model in Prolog.  */
/* The relational data model is based on a tabular            */
/* representation of data.                                      */
/* It allows you to establish two operations on relations :   */
/*     - A projection producing a new relationship               */
/*       Containing the specified columns of the original     */
/*       relation.                                            */
/*     - A join producing a new relationship by concatenating*/
/*       the end-to-end rows of the relation1 and relation2   */
/*       that have identical values  in the specified columns */
/*       by deleting the columns that repeat.                    */
/*                                                            */
/*****************************************************************/
```

```
/*****************************************************************/
/* Tests the equality of the type of two elements        */
/*                                                               */
/*****************************************************************/
sameType(X, Y) :- atom(X), atom(Y).
sameType(X, Y) :- integer(X), integer(Y).
sameType(X, Y) :- float(X), float(Y).

/*****************************************************************/
/* Tests whether an element is in a list                         */
/*****************************************************************/
member(X, [X|_]).
member(X, [_|Rest]) :- member(X, Rest).

/*****************************************************************/
/* Tests whether a list contains no repetitions                  */
/*****************************************************************/
noRepetition([]).
noRepetition([Elem|Rest]) :-
     \+ member(Elem, Rest), !,
     noRepetition(Rest).

/*****************************************************************/
/* Counts the number of elements in a list                       */
/*****************************************************************/
nbElements([], 0).
nbElements([_|Rest], N) :- nbElements(Rest, M), N is M + 1.

/*****************************************************************/
/* Tests whether two lists are the same length                   */
/*****************************************************************/
sameLength(List1, List2) :-
     nbElements(List1, N1),
     nbElements(List2, N2),
     N1=N2.

/*****************************************************************/
/* Verification compares the types of elements of two rows    */
/* of the table                                                  */
/*****************************************************************/
verifRows([], []).
verifRows([Elem1|Rest1], [Elem2|Rest2]) :-
     sameType(Elem1, Elem2), !,
     verifRows(Rest1,Rest2).

/*****************************************************************/
/* Verification of the table associated with a relation       */
/*****************************************************************/
verifTable([]).
```

```
verifTable([_]).
verifTable([Row1,Row2|Rest]) :-
     sameLength(Row1, Row2),
     verifRows(Row1, Row2),
     verifTable([Row2|Rest]).

/*****************************************************************/
/* Verification of the relation                                  */
/*****************************************************************/
verifRelation(relation(Header, [])) :- noRepetition(Header).
verifRelation(relation(Header, [Row|Rest])) :-
     noRepetition(Header),
     sameLength(Header, Row),
     verifTable([Row|Rest]).

/*****************************************************************/
/* Finds the index of an element in a list when it exists     */
/*  in the list                                                  */
/*****************************************************************/
index(Elem, [Elem|_], 1).
index(Elem, [_|Rest], Pos) :-
     index(Elem, Rest, Pos1),
     Pos is Pos1 + 1.

/*****************************************************************/
/* Finds the index of an element in a list, knowing that      */
/* it can not exist in the list                                    */
/*****************************************************************/
position(Elem, List, 0) :- \+ member(Elem, List), !.
position(Elem, List, Pos) :- index(Elem, List, Pos).

/*****************************************************************/
/* Finds the indices of projection headers                       */
/*****************************************************************/
findIndices([], _, []).
findIndices([Elem|RestE], List, [Index|RestI]) :-
     position(Elem, List, Index),
     findIndices(RestE, List, RestI).



/*********************************************************************/
/*Keeps the element index by the variable Index                      */
/*********************************************************************/
keepElt(Index, Index,[X|_], X).
keepElt(Index, Counter, [_|L], X) :-
             Counter1 is Counter + 1,
             keepElt(Index, Counter1, L, X).
```

```
/*************************************************************/
/* Keeps the elements indices by the list of indices         */
/* of a given row                                            */
/*************************************************************/
keepRow([],_,[]).
keepRow([I|Li], Rank, [X|Lr]) :-
             keepElt(I,1,Rank,X),
             keepRow(Li,Rank,Lr).


/*************************************************************/
/*  keeps all elements indices of the list indices (Li)      */
/*  of the table T                                           */
/*************************************************************/
keepAll(_,[],[]).
keepAll(Li, [Rank|T], [RankR|Tr]) :-
            keepRow(Li,Rank,RankR),
            keepAll(Li, T, Tr).

/*************************************************************/
/* Produces a new relation that contains only the specified  */
/* columns of the original relation                          */
/*************************************************************/
projection(HeaderP, relation(Header, Table), R) :-









/*************************************************************/
/* Concatenates two lists                                    */
/*************************************************************/
conc([], L, L).
conc([X|L1], L2, [X|L]) :- conc(L1, L2, L).


/*************************************************************/
/* Removes an element that is set to "Index" of the list     */
/*************************************************************/
remove(_,_,[],[]).
remove(Index, Index, [_|L], L) :- !.
remove(Index, Counter, [X|L], [X|Lr]) :-
```

```
            Counter1 is Counter + 1,
            remove(Index, Counter1, L, Lr).


/*****************************************************************/
/* Removes elements indices by the list of indices of         */
/* a given row.                                                  */
/*****************************************************************/
removeRow([], L, L).
removeRow([Index|Li], Row, RowR) :-
            remove(Index,1,Row, L),
            removeRow(Li, L, RowR).


/*****************************************************************/
/* Removes the elements indices by the list indices (Li)      */
/* of all the rows of a table                                     */
/*****************************************************************/
removeTable(_, [], []).
removeTable(Li, [Row|RestT], [RowR|RestTr]) :-
            removeRow(Li, Row, RowR),
            removeTable(Li, RestT,RestTr).


/*****************************************************************/
/* Removes elements indices from the list of indices (Li)     */
/* of the relation (relation (E, T))                              */
/*****************************************************************/
removeT(_,[],[]).
removeT(Li, relation(E, T), relation(Er, Tr)) :-
            removeRow(Li, E, Er),
            removeTable(Li, T, Tr).


/*****************************************************************/
/* Provides a list of pairs whose first element is the        */
/* position of an element of the table T1 and the second      */
/* element is the position of the same element in the table T2*/
/*****************************************************************/
correspondance2(_, [], _, []).
correspondance2(Counter, [X|RT1], T2, [[Counter,B]|RLci]) :-
            Counter1 is Counter + 1,
            position(X, T2, B),
            correspondance2(Counter1, RT1, T2, RLci).



/*****************************************************************/
/* Provides a list of pairs of indices corresponding          */
/* to the two tables T1 and T2                                    */
```

```
/*****************************************************************/
correspondance(relation(_,T1), relation(_, T2), Lci) :-
            correspondance2(1, T1, T2, Lci), !.


/*****************************************************************/
/* Searches the element at the position "Index"                  */
/*****************************************************************/
search(_, [], []).
search(1, [X|_], X).
search(Index, [_|RT], L) :- Y is Index - 1, search(Y, RT, L).


/*****************************************************************/
/* Constructs the final table                                    */
/*****************************************************************/
construct(_,_,[],[]).
construct(T1,T2,[[X,Y]|RestCi],[L|Res]) :-
             search(X,T1,L1),
             search(Y,T2,L2),
             conc(L1,L2,L),
             construct(T1,T2,RestCi,Res), !.


/*****************************************************************/
/* Joins two relations                                           */
/*****************************************************************/
join2(relation(E1,T1),relation(E2,T2),Li2,Lci,relation(E,T)) :-
            removeT(Li2,relation(E2,T2),relation(ER2, TR2)),
            conc(E1,ER2,E),
            construct(T1, TR2, Lci, T).


/*****************************************************************/
/* can join two relations from the list of indices of         */
/* correspondence getting a final relation                    */
/*****************************************************************/
join1(HeaderJ,Rel1,Rel2,Li2,RelR) :-
            projection(HeaderJ,Rel1,RelR1),
            projection(HeaderJ,Rel2,RelR2),
            correspondance(RelR1,RelR2,Lci),
            join2(Rel1,Rel2,Li2,Lci,RelR).




/*****************************************************************/
/* produces a new relation obtained by welding end-to-end     */
```

```
/* rows of relation1 and relation2 which have identical       */
/* values in the specified columns, taking care to remove     */
/* columns which are repeated                                 */
/************************************************************/
join(HeaderJ, Rel1,relation(ER2,Table2), RelR):-









/************************************************************/
/* Example of basic facts                                   */
/************************************************************/

/******************** Relation Registration *********************/
relation1(relation(['Name', 'Course'],
                   [['John', 'COMP232'],
                   ['Louise', 'COMP248'],
                   ['William', 'COMP348'],
                   ['William', 'COMP232']])).


/******************** Relation Assignment *********************/
relation2(relation(['Course', 'Prof', 'Local'],
                   [['COMP232', 'Tim W.', 'H6010'],
                   ['COMP248', 'Louise L.', 'H5605'],
                   ['COMP348', 'Mohamed T.', 'H7610']])).



/***************************** End of tests
*******************************/
```