Operatori Fuzzy

Andrea Gargarone

Giugno 2019

1 Operatori Fuzzy

Per poter lavorare con insiemi *fuzzy*, a partire da insiemi *crisp*, si è reso necessario introdurre nuovi operatori che permettono di definire predicati, generare insiemi fuzzy sulla base dei predicati definiti e effettuare varie operazioni tra di essi.
Il listato 1.1 mostra la struttura per la creazione di un nuovo operatore.

```
CREATE FUZZY OPERATOR operator_name
PARAMETERS (par_1 TYPE par_type, par_2 TYPE par_type, ...)
PRECONDITION conditions_on_parameters
EVALUATE expression | function(param)
RANGE (xMIN, xMAX)
POLYLINE (x1, y1), (x2, y2), .... (xn, yn)
```

Listato 1.1: Creazione di un operatore fuzzy

L'operatore CREATE FUZZY OPERATOR permette di creare un predicato linguistico grazie al quale è possibile generare il corrispondente insieme fuzzy. L'*operator_name* è il nome che lo identifica. Questo non deve necessariamente essere univoco, infatti è possibile avere più operatori con lo stesso nome. Ciò che ne permette la distinzione è il numero e il tipo dei parametri.
I PARAMETERS sono i parametri, elencati all'interno di parentesi tonde e separati tramite la virgola, i cui valori vengono valutati per il calcolo della membership. La membership è un valore compreso nell'intervallo compreso tra 0 e 1 e indica in che misura un oggetto appartiene all'insieme fuzzy. Ciascun

parametro deve essere seguito dall'istruzione TYPE, a sua volta seguita dal tipo del parametro. I tipi possono essere:

- SIMPLE: fields must be simple-value fields
- COMPLEX: fields must be complex (object) fields
- ARRAY: fields must be arrays
- STRING: fields must be strings
- NUMBER: fields must be numerical values
- INTEGER: fields must be integer values
- FLOAT: fields must be floating point values
- GEOMETRY: fields must be GeoJSON-compliant geometries
- FUZZY SET: fields must be numerical values between 0 and 1

FUZZY SET nuovo tipo!

L'istruzione PRECONDITION permette l'inserimento di tutte quelle condizioni necessarie a limitare il valore dei parametri così da impedire che assumano valori indesiderati.
L'operatore EVALUATE contiene l'espressione da valutare per il calcolo delle membership. L'*expression* è un'espressione contenente i parametri, operatori aritmetici e parentesi.
Il risultato ottenuto dall'espressione è poi valutato rispetto alla funzione di appartenenza definita tramite i punti indicati dopo il comando POLYLINE. Infatti questa funzione è una spezzata definita sul dominio rappresentato dal RANGE. Il suo codominio è l'intervallo continuo compreso tra 0 e 1. Questi valori rappresentano il grado di appartenenza all'insieme fuzzy definito. Il valore 0 indica la non appartenenza mentre il valore 1 indica la piena appartenenza. Tutti i valori intermedi indicano in quale misura l'oggetto in questione appartiene all'insieme fuzzy. Nel caso in cui i valori sui quali viene calcolata la membership sono al di fuori del RANGE, la membership assegnata all'elemento è pari a 0.

Come è possibile notare, nell'operatore EVALUATE si ha una doppia possibilità. La prima, appena discussa, è l'utilizzo di una *expression*. La seconda,

invece, consiste nel richiamare una funzione alla quale vengono passati, tra parentesi tonde, i parametri necessari per il calcolo della membership. Ciò è utile ogni qualvolta il calcolo della membership necessita di operazioni complesse, non realizzabili nel linguaggio J-CO-QL.
Per la definizione e l'implementazione della funzione si rimanda al paragrafo 4.

```
CREATE FUZZY OPERATOR small
PARAMETERS (number_of_rooms TYPE INTEGER)
PRECONDITION number_of_rooms >= 1
EVALUATE number_of_rooms
RANGE (1, 40)
POLYLINE (1, 1), (15, 1), (40, 0)
```

Esempio 1.1: Creazione dell'operatore fuzzy *small*

Come si può notare la valutazione del grado di appartenenza all'insieme fuzzy viene effettuata solo sul parametro *.number_ of_ rooms* che è di tipo INTEGER. Inoltre la PRECONDITION specifica che tale valore deve necessariamente essere maggiore o uguale a 1.
La valutazione avviene sul parametro indicato all'interno di un RANGE compreso tra 1 e 40. Valori al di fuori di questo intervallo forniscono come valore di membership 0.

Con CREATE FUZZY OPERATOR è anche possibile creare predicati complessi, che permettono, ad esempio, l'aggregazione di più insiemi fuzzy a ciascuno dei quali può essere attribuito un peso differente.

```
CREATE FUZZY OPERATOR wag
PARAMETERS (weight TYPE NUMBER, fs1 TYPE FUZZY SET,
            fs2 TYPE FUZZY SET)
PRECONDITION weight >= 0 AND weight <= 1
EVALUATE weight * MEMBERSHIP(fs1) +
         (1 - weight) * MEMBERSHIP(fs2)
```

Esempio 1.2: Creazione dell'operatore wag per aggregare due insiemi fuzzy con pesi weight e 1-weght

In questo caso, oltre ai valori di membership relativi ai due insiemi fuzzy (di tipo FUZZY SET), viene anche passato il parametro intero *weight* che rappresenta il peso da associare associato all'insieme fuzzy.
Nella PRECONDITION si verifica che il parametro *weight* sia compreso tra 0 e 1. Nell'EVALUATE si trova l'*expression* che valuta il valore di membership complessivo derivato dall'aggregazione dei due insiemi fuzzy.
Come già detto in precedenza, anche in questo caso è possibile utilizzare, al posto dell'*expression*, una funzione. Si veda la Sezione 4.
Quando RANGE e POLYLINE sono omessi, come nell'Esempio 1.2, significa che questi assumono i valori di default. In particolare si ha RANGE(0,1) e POLYLINE (0,0), (1,1).

2 Generazione di un insieme Fuzzy

Definito l'operatore fuzzy, si può creare l'insieme fuzzy. La generazione dell'insieme avviene mediante l'operatore FILTER che permette di lavorare solo sugli oggetti di interesse.

```
GET COLLECTION collection@db
FILTER
    CASE
        WHERE ...
        GENERATE FUZZY SET fuzzyset_name
        EVALUATE fuzzy_expression
DROP OTHERS
SET INTERMEDIATE ...
```

Listato 2.1: Codice per la generazione di un insieme fuzzy

Con FILTER si selezionano solamente gli oggetti che rispettano le condizioni dichiarate nel ramo WHERE.
L'operatore GENERATE FUZZY SET genera l'insieme e gli assegna un nome univoco. L'operatore EVALUATE contiene una *fuzzy expression* che è responsabile della valutazione della membership.

Generato l'insieme fuzzy, la nuova collezione salvata conterrà tutti gli elementi della collezione di partenza per i quali è stato possibile valutare la membership. A ciascuno di questi elementi viene quindi aggiunto un nuovo campo, "~fuzzysets", contenente i valori delle membership di tutti gli insiemi fuzzy a cui l'oggetto appartiene, come mostrato nel seguito:

```
{
    "field_1'': "value_1'',
    "field_2'': "value_2'',
    ...
    "~fuzzysets'': {
        "fuzzyset_name'': value,
        ...
```

```
    }
}
```

Si prenda in considerazione la collezione *hotels* così composta:

```
{
    "name": "Stella",
    "star": 2,
    "number_of_rooms": 23,
    "price": 40,
    "distance_from_center": 3000
},
{
    "name": "Central",
    "star": 3,
    "number_of_rooms": 28,
    "price": 70,
    "distance_from_center": 1000
},
{
    "name": "Rosengarten",
    "star": 4,
    "number_of_rooms": 35,
    "price": 150,
    "distance_from_center": 200
},
{
    "name": "Moseralm",
    "star": 5,
    "number_of_rooms": 45,
    "price": 190,
    "distance_from_center": 50
},
{
    "name": "Diana",
    "star": 3,
    "number_of_rooms": 30,
```

```
    "price": 60,
    "distance_from_center": 1380
},
{
    "name": "Tyrol",
    "star": 3,
    "number_of_rooms": -1,
    "price": 65,
    "distance_from_center": 1200
}
```

Partendo dalla collezione *hotels* si vuole creare l'insieme fuzzy *small_ hotels* utilizzando l'operatore *small* definito precedentemente.

```
GET COLLECTION hotels@db
FILTER
    CASE
        WHERE WITH .number_of_rooms
        GENERATE FUZZY SET small_hotels
        EVALUATE IF-FAILS(small(number_of_rooms), 0)
DROP OTHERS
SET INTERMEDIATE AS small_hotels
```

Esempio 2.1: generazione dell'insieme fuzzy *small_ hotels* utilizzando il predicato *small*

Con il comando GET COLLECTION la collezione *hotels*, presente nel database, diventa la nuova *temporary collection*. L'operatore FILTER filtra gli elementi della *tc* in accordo con la condizione WHERE. Questa specifica che si devono prendere in considerazione solamente gli elementi che possiedono l'attributo *"room_ number"*.
L'operatore GENERATE FUZZY SET crea l'insieme fuzzy di nome *small_ hotels*. La membership di ciascun oggetto viene calcolata attraverso l'operatore *small*, sul parametro *number_ of_ rooms*.
Con DROP OTHERS si eliminano tutti gli elementi per i quali non è stato possibile il calcolo della membership a causa la mancanza del campo relativo al

parametro richiesto.
Infine l'operatore SET INTERMEDIATE salva la collezione temporanea ottenuta nel *Intermediate Result Database* con il nome *small_ hotels*.

Di seguito è riportata la collezione *small_ hotels* dove è possibile vedere le membership di ciascun oggetto. Si può notare che a ciascun elemento è stata associata la membership relativa al predicato definito, anche per gli oggetti il cui valore di membership è pari a 0. Questo avviene perchè il valore del parametro è fuori dal RANGE o perchè la funzione di appartenenza ha dato come risultato una membership nulla.
Inoltre si nota che l'ultimo oggetto (hotel Tyrol), avendo il campo *number_ of_ rooms* di valore pari a -1, non rispetta la precondizione. Tuttavia grazie alla funzione IF-FAILS viene attribuito come membership il valore 0.

Di seguito è riportato l'insieme fuzzy *small_ hotels*.


```
{
    "name": "Stella",
    "star": 2,
    "number_of_rooms": 23,
    "price": 40,
    "distance_from_center": 3000,
    "~fuzzysets": {
        "small_hotels": 0.6
    }
},
{
    "name": "Central",
    "star": 3,
    "number_of_rooms": 28,
    "price": 70,
    "distance_from_center": 1000,
    "~fuzzysets": {
        "small_hotels": 0.4,
    }
},
{
```

```
    "name": "Rosengarten",
    "star": 4,
    "number_of_rooms": 35,
    "price": 150,
    "distance_from_center": 200,
    "~fuzzysets": {
        "small_hotels": 0.2
    }
},
{
    "name": "Moseralm",
    "star": 5,
    "number_of_rooms": 45,
    "price": 190,
    "distance_from_center": 50,
    "~fuzzysets": {
        "small_hotels": 0
    }
},
{
    "name": "Diana",
    "star": 3,
    "number_of_rooms": 30,
    "price": 60,
    "distance_from_center": 1380,
    "~fuzzysets": {
        "small_hotels": 0.4
    }
},
{
    "name": "Tyrol",
    "star": 3,
    "number_of_rooms": -1,
    "price": 65,
    "distance_from_center": 1200,
    "~fuzzysets": {
        "small_hotels": 0
```

```
    }
}
```

Ora si applica un nuovo operatore *cheap* all'insieme appena generato *small_ hotel* per generare un nuovo insieme fuzzy.

```
CREATE FUZZY OPERATOR cheap
PARAMETERS (price TYPE INTEGER)
PRECONDITION price > 0
EVALUATE price
RANGE (1, 100)
POLYLINE (0, 1), (50, 1), (100, 0)

GET COLLECTION small_hotels
FILTER
    CASE
        WHERE WITH price
        GENERATE FUZZY SET cheap_hotels
        EVALUATE IF-FAILS(cheap(price), 0)
    KEEP OTHERS
SET INTERMEDIATE AS small_cheap_hotels
```

Esempio 2.2: Creazione del predicato *cheap* e generazione dell'insieme fuzzy *small_ cheap_ hotels*

Si ottiene il seguente risultato:

```
{
    "name": "Stella",
    "star": 2,
    "number_of_rooms": 23,
    "price": 40,
    "distance_from_center": 3000,
    "~fuzzysets": {
        "small_hotels": 0.6,
        "cheap_hotels": 1
```

```
    }
},
{
    "name": "Central",
    "star": 3,
    "number_of_rooms": 28,
    "price": 70,
    "distance_from_center": 1000,
    "~fuzzysets": {
        "small_hotels": 0.4,
        "cheap_hotels": 0.6
    }
},
{
    "name": "Rosengarten",
    "star": 4,
    "number_of_rooms": 35,
    "price": 150,
    "distance_from_center": 200,
    "~fuzzysets": {
        "small_hotels": 0.2,
        "cheap_hotels": 0
    }
},
{
    "name": "Moseralm",
    "star": 5,
    "number_of_rooms": 45,
    "price": 190,
    "distance_from_center": 50,
    "~fuzzysets": {
        "small_hotels": 0,
        "cheap_hotels": 0
    }
},
{
    "name": "Diana",
```

```
    "star": 3,
    "number_of_rooms": 30,
    "price": 60,
    "distance_from_center": 1380,
    "~fuzzysets": {
        "small_hotels": 0.4,
        "cheap_hotels": 0.8
    }
},
{
    "name": "Tyrol",
    "star": 3,
    "number_of_rooms": -1,
    "price": 65,
    "distance_from_center": 1200,
    "city": "Nova Levante",
    "~fuzzysets": {
        "small_hotels": 0,
        "cheap_hotels": 0.7,
    }
}
```

Ora si vogliono aggregare i due insiemi fuzzy assegnando a ciascuno di essi un peso utilizziamo l'operatore *wag* definito precedentemente.

```
GET COLLECTION small_cheap_hotels
FILTER
CASE
    WHERE WITH
    GENERATE FUZZY SET small_and_cheap_hotels
    EVALUATE IF-FAILS(wag(0.3, small_hotels, cheap_hotels), 0)
DROP OTHERS
SET INTERMEDIATE AS small_and_cheap_hotels
```

Esempio 2.3: Aggregazione dei due insiemi fuzzy *small_hotels* e *in_city_center*

Il risultato che si ottiene è il seguente:


```
{
    "name": "Stella",
    "star": 2,
    "number_of_rooms": 23,
    "price": 40,
    "distance_from_center": 3000,
    "~fuzzysets": {
        "small_hotels": 0.6,
        "cheap_hotels": 1,
        "small_and_cheap_hotels": 0.8
    }
},
{
    "name": "Central",
    "star": 3,
    "number_of_rooms": 28,
    "price": 70,
    "distance_from_center": 1000,
    "~fuzzysets": {
        "small_hotels": 0.4,
        "cheap_hotels": 0.6,
        "small_and_cheap_hotels": 0.5
    }
},
{
    "name": "Rosengarten",
    "star": 4,
    "number_of_rooms": 35,
    "price": 150,
    "distance_from_center": 200,
    "~fuzzysets": {
        "small_hotels": 0.2,
        "cheap_hotels": 0,
        "small_and_cheap_hotels": 0.06
    }
```

```
},
{
    "name": "Moseralm",
    "star": 5,
    "number_of_rooms": 45,
    "price": 190,
    "distance_from_center": 50,
    "~fuzzysets": {
        "small_hotels": 0,
        "cheap_hotels": 0,
        "small_and_cheap_hotels": 0
    }
},
{
    "name": "Diana",
    "star": 3,
    "number_of_rooms": 30,
    "price": 60,
    "distance_from_center": 1380,
    "~fuzzysets": {
        "small_hotels": 0.4,
        "cheap_hotels": 0.8,
        "small_and_cheap_hotels": 0.6
    }
},
{
    "name": "Tyrol",
    "star": 3,
    "number_of_rooms": -1,
    "price": 65,
    "distance_from_center": 1200,
    "city": "Nova Levante",
    "~fuzzysets": {
        "small_hotels": 0,
        "cheap_hotels": 0.7,
        "small_and_cheap_hotels": 0.4
    }
```

```
}
```

Così si è ottenuto il valore di membership relativo all'aggregazione di sue insiemi fuzzy pesando con pesi diversi i valori di appartenenza ai singoli insiemi.

3 Query su insiemi Fuzzy

Definito il processo di generazione di insiemi fuzzy, nel seguito è proposto un esempio di query su una collezione fuzzy.

Si prenda in cosiderazione il risultato dell'Esempio 2.1. Si vuole estrarre dall'insieme fuzzy *small_hotels* tutti gli hotel che soddisfano il predicato *small* con un grado di appartenenza maggiore o uguale a 0.5. Per fare ciò si utilizza l'operatore FILTER, dove la condizione WHERE valuta l'appartenenza dell'oggetto all'insieme fuzzy mediante il predicato WITHIN FUZZY SET seguito dal nome dell'insieme fuzzy. Così facendo vengono presi in considerazione solamente gli oggetti che appartengono all'insieme fuzzy desiderato. L'istruzione ALPHA-CUT indica il valore minimo che la membership di ciascun elemento deve avere per poter essere presa in considerazione. La preposizione ON specifica su quale attributo all'interno del campo *"~fuzzysets"* si deve effettuare la valutazione della membership. Nella *temporary collection* saranno contenuti quindi solamente gli oggetti con membership maggiore o uguale al valore specificato nell'istruzione ALPHA-CUT.
Opzionalmente è possibile utilizzare l'operatore DROPPING FUZZY SETS seguito dal nome di uno o più insiemi fuzzy. Questo comando elimina i valori delle membership associate allo specifico insieme fuzzy.

```
GET COLLECTION small_hotels
FILTER
    CASE
        WHERE WITHIN FUZZY SETS small_hotels
        ALPHA-CUT 0.5 ON small_hotels
        DROPPING FUZZY SETS small_hotels
DROP OTHERS
```

Esempio 3.1: Query sull'insieme fuzzy *small_hotels*

Di seguito il risultato della query:

```
{
    "name": "Stella",
```

```
    "star": 2,
    "number_of_rooms": 23,
    "price": 40,
    "distance_from_center": 3000
}
```

Solo l'hotel "Stella" possiede una membership relativa al predicato *small* maggiore o uguale a 0.5. Inoltre con l'operatore DROPPING FUZZY SETS è stato eliminato il campo *"small_hotels": 0.6*. In questo modo è stata attuata una defuzzyficazione, infatti l'oggetto risultante è *crisp*.

L'esempio precedente riporta una query semplice su un insieme fuzzy. Si consideri ora un esempio in cui, come spesso accade, si ha la necessità di lavorare con collezioni eterogenee. Si supponga quindi di avere una collezione, *merge_collection*, ottenuta tramite l'operazione MERGE di due collezioni. La prima è costituita da oggetti per i quali sono stati valutati differenti predicati fuzzy. In particolare nell'esempio la collezione contiene i valori di membership relativi agli insiemi *small_hotels* e *in_city_center* (generato attraverso l'operatore *close_to_center*, Listato 3.1). La seconda invece contiene oggetti per i quali è stato valutato un solo predicato fuzzy: *small_hotels*. La mancanza, in alcuni oggetti, dei valori di membership può significare che la funzione di appartenenza è stata valutata e ha prodotto come risultato un valore nullo. Oppure potrebbe non essere stata valutata. Quindi come capire in quale dei due casi ci si trova? Per ovviare a ciò sono stati introdotti due nuovi predicati: KNOWN e UNKNOWN. Il primo permette di selezionare tutti gli oggetti che possiedono il valore di membership relativo all'insieme fuzzy *small_hotels*, qualunque esso sia. Infatti, nell'esempio si è sicuri che il valore di membership per l'insieme *small_hotels* è stato valutato per ciascun oggetto. Invece, l'operatore UNKNOWN permette, per gli oggetti che non possiedono il valore di membership relativo all'insieme *in_city_center*, di calcolarlo. Così facendo si ha la certezza che il grado di appartenenza all'insieme è stato valutato per tutti gli elementi della collezione.

```
CREATE FUZZY OPERATOR close_to_center
PARAMETERS (distance_from_center)
PRECONDITION distance_from_center > 0
EVALUATE distance_from_center
RANGE (0, 2000)
POLYLINE (0, 1), (1000, 1), (2000, 0)
```

Listato 3.1: Operatore fuzzy *close_to_center*

Di seguito è riportata la collezione *merge_collection* di partenza su cui eseguire la query. Come è possibile notare alcuni oggetti non possiedono i valori di membership di entrambi gli insiemi fuzzy considerati.


```
{
    "name": "Stella",
    "star": 2,
    "number_of_rooms": 23,
    "price": 40,
    "distance_from_center": 3000,
    "~fuzzysets": {
        "small_hotels": 0.6,
        "in_city_center": 0
    }
},
{
    "name": "Central",
    "star": 3,
    "number_of_rooms": 28,
    "price": 70,
    "distance_from_center": 1000,
    "~fuzzysets": {
        "small_hotels": 0.4,
    }
},
{
```

```
    "name": "Rosengarten",
    "star": 4,
    "number_of_rooms": 35,
    "price": 150,
    "distance_from_center": 200,
    "~fuzzysets": {
        "small_hotels": 0.2,
    }
},
{
    "name": "Moseralm",
    "star": 5,
    "number_of_rooms": 45,
    "price": 190,
    "distance_from_center": 50
    "~fuzzysets": {
        "small_hotels": 0,
        "in_city_center": 0.6
    }
},
{
    "name": "Diana",
    "star": 3,
    "number_of_rooms": 30,
    "price": 60,
    "distance_from_center": 1380,
    "~fuzzysets": {
        "small_hotels": 0.4,
        "in_city_center": 0.6
    }
},
{
    "name": "Tyrol",
    "star": 3,
    "number_of_rooms": -1,
    "price": 65,
    "distance_from_center": 1200,
```

```
    "city": "Nova Levante",
    "~fuzzysets": {
        "small_hotels": 0,
        "in_city_center": 0.8
    }
}
```

Ora si vuole creare una query per ottenere tutti gli oggetti che appartengono ad entrambi gli insiemi fuzzy con un valore di ALPHA-CUT pari a 0.4. Siccome non tutti gli oggetti possiedono entrambe le membership su cui si va a effettuare la valutazione e siccome non si può sapere a priori nulla sulla valutazione della membership *in_ city_ center*, entrano in gioco gli operatori KNOWN e UNKNOWN.

```
GET COLLECTION merge_collection@db
FILTER
CASE
    WHERE
        KNOWN FUZZY SET small_hotels
        AND
        UNKNOWN FUZZY SET in_city_center
        GENERATE FUZZY SET in_city_center
        EVALUATE
            IF-FAILS(close_to_center(.distance_from_center),0)
    GENERATE FUZZY SET wanted
    EVALUATE small_hotels AND in_city_center
    ALPHA-CUT 0.4 ON wanted
    DROPPING small_hotels, in_city_center
KEEP OTHERS
SET INTERMEDIATE AS small_and_in_center_hotels
```

Di seguito il risultato ottenuto dopo l'esecuzione della query:

```
{
```

```
    "name": "Central",
    "star": 3,
    "number_of_rooms": 28,
    "price": 70,
    "distance_from_center": 1000,
    "~fuzzysets": {
        "wanted": 0.4
    }
},
{
    "name": "Diana",
    "star": 3,
    "number_of_rooms": 30,
    "price": 60,
    "distance_from_center": 1380,
    "~fuzzysets": {
        "wanted": 0.4
    }
}
```

4 Funzioni JavaScript

Come già accennato nella Sezione 1 in alcuni casi può essere utile definire funzioni, anche complesse, per il calcolo del valore sul quale viene valutata la membership. Essendo J-CO-QL è un linguaggio dichiarativo di interrogazione nello stile SQL, non è un linguaggio adatto all'implementazione di funzioni. Per questo motivo è necessario appoggiarsi ad un linguaggio di programmazione esterno. Il linguaggio scelto è JavaScript.

```
CREATE JAVASCRIPT FUNCTION function_name(parameters)
PARAMETERS (...)
PRECONDITION ...
BODY
{
    // istruzioni JavaScript
    return ...
}
END BODY
```

L'operatore CREATE JAVASCRIPT FUNCTION dichiara la funzione e i parametri che devono essere passati. I PARAMETERS sono i parametri necessari per il calcolo del valore di membership. La PRECONDITION contiene delle condizioni che devono essere rispettate dai parametri per evitare che assumano valori indesiderati. All'interno dei tag BODY e END BODY, tra parentesi graffe, si inserisce il codice JavaScript per il calcolo della funzione richiesta. Come ultima istruzione JavaScript deve esserci una *return* che restituisce il valore dell'attributo richiesto, usato poi per il calcolo della membership.

Come anticipato nella Sezione 1 un esempio è il calcolo della distanza geodetica tra due punti, dei quali si conoscono solamente le coordinate geografiche. In questo caso ci si deve affidare ad un linguaggio di programmazione esterno per l'implementazione della funzione che calcoli e restituisca il valore della distanza.

Si supponga di avere la collezione contenente coppie di luoghi dei quali si vuole calcolare la distanza geodetica.

```
CREATE JAVASCRIPT FUNCTION geodetic_distance(lat1, lon1,
                                             lat2, lon2)
PARAMETERS (lat1 TYPE FLOAT, lon1 TYPE FLOAT,
            lat2 TYPE FLOAT, lon2 TYPE FLOAT)
PRECONDITION lat1 >= -90.0 AND lat2 >= -90.0 AND
             lat1 <= 90.0 AND lat2 <= 90.0 AND
             lon1 >= -180.0 AND lon2 >= -180.0 AND
             lon1 <= 180.0 AND lon2 <= 180.0
BODY
{
    /*
     * codice JavaScript per il calcolo
     * della distanza geodetica
     *
     */
    return distance;
}
END BODY
```

L'operatore fuzzy in questo caso è definito come segue:

```
CREATE FUZZY OPERATOR g_distance
PARAMETERS (lat1 TYPE FLOAT, lon1 TYPE FLOAT,
            lat2 TYPE FLOAT, lon2 TYPE FLOAT)
PRECONDITION lat1 >= -90.0 AND lat2 >= -90.0 AND
              lat1 <= 90.0 AND lat2 <= 90.0 AND
              lon1 >= -180.0 AND lon2 >= -180.0 AND
              lon1 <= 180.0 AND lon2 <= 180.0
EVALUATE geodetic_distance(lat1, lon1, lat2, lon2)
RANGE (0,20)
POLYLINE (0,1),(10,1),(20,0)
```

Sempre nella Sezione 1, si fa riferimento ad un'altra possibilità di impiego di funzioni JavaScript. Cioè uella di poter dichiarare funzioni personalizzate per aggregare secondo differenti criteri due o più insiemi fuzzy.

Lo stesso risultato dell' Esempio 2.3 può essere realizzato utilizzando una funzione JavaScript.

```
CREATE JAVASCRIPT FUNCTION wag_03(small, cheap)
PARAMETERS (small TYPE FUZZY SET, cheap TYPE FUZZY SET)
PRECONDITION small >= 0 AND small <= 1 AND
             cheap >= 0 AND cheap <= 1
BODY
{
    return ( 0.3 * small + 0.7 * in_center );
}
END BODY
```

Questa funzione viene quindi richiamata all'interno del predicato:

```
CREATE FUZZY OPERATOR aggregate_with_priority_small_cheap
PARAMETERS (small_hotels TYPE FUZZY SET,
            cheap_hotels TYPE FUZZY SET)
PRECONDITION small_hotels >= 0 AND small_hotels <= 1 AND
             cheap_hotels >= 0 AND cheap_hotels <= 1
EVALUATE wag_03(small_hotels, cheap_hotels)
```

Quindi è possibile applicare l'operatore appena creato alla collezione *small_ cheap_ hotels* e ottenere così lo stesso risultato dell'Esempio 2.3, cioè la collezione *small_ and_ cheap_ hotels*, utilizzando la funzione *wag_ 03* al posto dell'*expression*.

5 Join con insiemi Fuzzy

Quando si effettua un'operazione di JOIN tra due insiemi fuzzy è necessario gestire il campo "~fuzzysets". Per fare ciò è stata introdotta una clausola: SET FUZZY SET. Questa ggiunta è un'estensione della struttura del normale operatore JOIN.

```
JOIN OF COLLECTIONS C1, C2
SET FUZZY SETS C1.fs_1, C1.fs_2 AS fs_name,
               C1.fs_3 AND c2.fs_2 AS fs
DROPPING SOURCE FUZZY SETS | KEEPING SOURCE FUZZY SETS
```

Listato 5.1: JOIN con estensione Fuzzy

L'aggiunta della clausola SET FUZZY SETS permette quindi di selezionare tutti gli insiemi fuzzy da considerare nella collezione risultante a seguito dell'esecuzione dell'operazione di JOIN (Listato 5.1). Nel caso di insiemi fuzzy con lo stesso nome il comando AS seguito dal nuovo nome permette di cambiare il nome dell'insieme. Se, in presenza di ambiguità, gli insiemi non vengono rinominati allora si prende in considerazione solo quello con valore di membership più basso.
Oltre ad elencare gli attributi fuzzy è possibile anche aggregarli mediante le congiunzioni AND e OR, come mostrato nel Listato 5.1.
Una volta selezionati gli insiemi fuzzy voluti, il comando
KEEPING SOURCE FUZZY SETS fa si che tutti gli attributi fuzzy presenti in entrambe le collezioni saranno contenuti nella collezione risultato.
In alternativa è possibile utilizzare il comando DROPPING SOURCE FUZZY SETS per eliminare tutti gli insiemi fuzzy tranne quelli che sono espressamente dichiarati all'interno della clausola SET FUZZY SETS.

ESEMPIO:
Supponiamo di avere la collezione *hotels* sulla quale sono stati calcolati gli insiemi fuzzy *small_hotels*, *in_city_center*, e *cheap_hotels*

```
{
    "name": "Stella",
    "star": 2,
```

```
    "number_of_rooms": 23,
    "price": 40,
    "distance_from_center": 3000,
    "city": "Nova Levante",
    "~fuzzysets": {
        "small_hotels": 0.6,
        "cheap_hotels": 1,
        "in_city_center": 0
    }
},
{
    "name": "Central",
    "star": 3,
    "number_of_rooms": 28,
    "price": 70,
    "distance_from_center": 1000,
    "city": "Nova Levante",
    "~fuzzysets": {
        "small_hotels": 0.4,
        "cheap_hotels": 0.6,
        "in_city_center": 0.4
    }
},
.....
```


e la collezione *cities* sulla quale sono stati calcolati gli insiemy fuzzy *small_ city* e *small_ density*.


```
{
    "name": "Nova Levante",
    "inhabitants": 1945,
    "surface_extension": 51.1,
    "density": 38.06,
    "state": "Italia",
    "region": "Trentino - Alto Adige",
    "province": "Bolzano",
    "~fuzzysets": {
        "small_cities": 0.8,
```

```
        "small_density": 1
    }
}, ......
```

Si applica ora l'operatore di JOIN così definito:

```
JOIN OF COLLECTIONS hotels AS h, cities AS c
SET FUZZY SETS h.small_hotels, h.cheap_hotels, c.small_cities
DROPPING SOURCE FUZZY SETS
CASE
    WHERE WITH c.name = h.city
    GENERATE { name: .h.name,
               star: .h.star,
               number_of_rooms: .h.number_of_rooms,
               price: .h.price,
               distance_from_center: .h.distance_from_center,
               city: h.city,
               inhabitants: .c.inhabitants,
               surface_extension: .c.surface_extension }
DROP OTHERS
SET INTERMEDIATE ....
```

Esempio 5.1: Join tra le due due collezioni fuzzy *hotels* e *cities*

La collezione risultante contiene quindi elementi così strutturati:

```
{
    "name": "Stella",
    "star": 2,
    "number_of_rooms": 23,
    "price": 40,
    "distance_from_center": 3000,
    "city": "Nova Levante",
    "inhabitants": 1945,
    "surface_extension": 51.1,
    "~fuzzysets": {
        "small_hotels": 0.6,
```

```
        "cheap_hotels": 1,
        "small_cities": 0.8
    }
}, .....
```

Per questioni di spazio qui è riportato un solo elemento della collezione risultato.

5.1 Spatial Fuzzy Join

Per poter valutare le *geometry* in maniera fuzzy è stata aggiunta la clausola SET FUZZY SETS anche all'operatore SPATIAL JOIN. In questo caso la clausola permette la creazione di insiemi fuzzy basati sulle geometrie. Per poter fare ciò sono stati definiti alcuni predicati fuzzy e alcune nuove funzioni.

I predicati fuzzy, di seguito riportati, permettono il calcolo del valore di membership:

INSIDE(side) Predicato fuzzy per valutare l'inclusione di una geometria nell'altra. Il *side* può assumere valori *LEFT* o *RIGHT* e permette di indicare quale geometria include l'altra. Nel caso di *side = LEFT* il calcolo dell'inclusione è dato da: *Area(Intersezione(g1, g2)/Area(g1)*. Nel caso di *side = RIGHT* il calcolo è dato da: *Area(Intersezione(g1, g2)/Area(g2)*.
Il risultato rappresenta il valore di membership relativo all'inclusione.

OVERLAP Predicato fuzzy per valutare la sovrapposizione di due geometrie. Il calcolo della membership è dato da *Area(Intersezione(g1, g2)) / Area(Unione(g1, g2))*.

HOW-MEET(side) Predicato fuzzy per valutare se due geometry condividono parte dei loro confini e in misura. L'operazione per il calcolo della membership è la seguente: *l_comune / 2p(g_left)* (*l_comune / 2p(g_right)* nel caso contrario) e consiste nel calcolo della lunghezza della parte in comune rapportata al perimetro della geometria indicata dal parametro *side*.

Di seguito sono invece riportate le nuove funzioni definite in aggiunta a quelle già esistenti nel linguaggio J-Co. Il valore da loro restituito è poi impiegato per il calcolo dei valori di membership grazie ad operatori fuzzy definiti dall'utente.

DIRECTION(from) Funzione crisp per il calcolo della direzione rispetto ai punti cardinali. Se *from* = *LEFT* indica la direzione del vettore spaziale tra il centro della geometria di sinistra e il centro della geometria di destra. Se *from* = *RIGHT* vale il viceversa. Il calcolo restituisce un valore tra 0 e 360 che può poi essere valutato attraverso predicati fuzzy definiti dall'utente per il calcolo del valore di membership.

AREA(unit, geometry) Funzione crisp che restituisce il valore dell'area della geometria indicata dal parametro *geometry* nell'unità definita dal parametro *unit*. Le unità di misura possono essere M (metri quadri), KM (chilometri quadri) o ML (miglia quadri). AREA(unit, geometry) != AREA(unit) –> ridefinire AREA(unit) come COMMON AREA(unit)???

PERIMETER(unit, geometry) Funzione crisp che restituisce il valore del perimetro della geometria indicata dal parametro *geometry*. L'unità di misura può essere M (metri), KM (chilometri) o ML (miglia).

```
SPATIAL JOIN OF COLLECTIONS C1, C2
ON .....
SET GEOMETRY .....
SET FUZZY SETS name_fuzzy_set
EVALUATE fuzzy_expression
KEEPING SOURCE FUZZY SETS | DROPPING SOURCE FUZZY SETS
CASE
    WHERE WITH ....
```

Listato 5.2: SPATIAL JOIN con predicati fuzzy

Come mostrato nel Listato 5.2, la sintassi dell'operatore SPATIAL JOIN non è stata modificata. L'unica novità è l'aggiunta della clausola SET FUZZY SETS, seguita dal nome dell'insieme fuzzy che si vuole generare. La *fuzzy_ expression*, che segue l'istruzione EVALUATE, è costituita o dal nome di un predicato fuzzy standard o da un operatore fuzzy definito dall'utente al quale vengono passati come parametri i valori restituiti da una o più funzioni.
Nel primo caso si avrebbe ad esempio:

```
SET FUZZY SETS name_of_fuzzy_set
EVALUATE OVERLAP
```

Mentre nel secondo caso:

```
SET FUZZY SETS name_of_fuzzy_set
EVALUATE fuzzy_operator(DISTANCE(KM))
```

Visto che le collezioni che entrano in gioco nello SPATIAL JOIN possono essere fuzzy, anche in questo caso è possibile scegliere se mantenere tutti gli insiemi fuzzy delle collezioni originarie o eliminarli dalla collezione risultato grazie al comando KEEPPING SOURCE FUZZY SETS o DROPPING SOURCE FUZZY SETS.

ESEMPIO: Si supponga di avere la collezione crisp *City*

```
{
    "City": "Bergamo",
    "Region": "Lombardia",
    "State": "Italy",
    .....
    "~geometry": {
        "type": "Point",
        "coordinates": [9.67, 45.6988889]
    }
},
.....
```

e la collezione crisp *Tweet*

```
{
    "tweetID": 1,
    "~geometry": {
        "coordinates": [9.69988419, 45.66594707],
        "type": "Point"
    },
},
{
    "tweetID": 2,
    "~geometry": {
        "coordinates": [9.69930355, 45.66561062],
        "type": "Point"
    },
```

```
},
{
    "tweetID": 3,
    "~geometry": {
        "coordinates": [9.69946617, 45.66563574],
        "type": "Point"
    },
},
{
    "tweetID": 4,
    "~geometry": {
        "coordinates": [9.69950141, 45.66532037],
        "type": "Point"
    },
},
{
    "tweetID": 5,
    "~geometry": {
        "coordinates": [9.69946387, 45.66533018],
        "type": "Point"
    },
},
{
    "tweetID": 6,
    "~geometry": {
        "coordinates": [9.69959869, 45.66584253],
        "type": "Point"
    },
},
{
    "tweetID": 7,
    "~geometry": {
        "coordinates": [9.69543276, 45.66445455],
        "type": "Point"
    },
},
{
```

```
    "tweetID": 8,
    "~geometry": {
        "coordinates": [9.6992909, 45.6657278],
        "type": "Point"
    },
},
{
    "tweetID": 9,
    "~geometry": {
        "coordinates": [9.69980096, 45.6633577],
        "type": "Point"
    },
},
{
    "tweetID": 10,
    "~geometry": {
        "coordinates": [9.69665857, 45.66715451],
        "type": "Point"
    },

},
.....
```

Si vuole effettuare uno SPATIAL JOIN per vedere quali Tweet sono stati pubblicati nelle vicinanze del centro della città di Bergamo.

Nel Esempio 5.1, lo SPATIAL JOIN viene effettuato tra le due collezioni *City* e *Tweet*. Si indica la condizione spaziale di JOIN e si definisce la geometria del nuovo oggetto risultante dall'operazione di JOIN.
Dopo la clausola SET FUZZY SETS viene indicato il nome del l'insieme fuzzy. La distanza tra il centro città e la posizione nella quale è stato generato il tweet viene valutata poi attraverso una funzione per il calcolo della membership definita grazie all'operatore fuzzy *distance_ operator* (Listato 5.3) al quale viene passato come parametro il valore di ritorno della funzione DISTANCE.

```
SPATIAL JOIN OF COLLECTIONS City AS c, Tweet AS t
ON DISTANCE(M) <= 1200
SET GEOMETRY RIGHT
SET FUZZY SETS distance_from_city_center
EVALUATE distance_operator(DISTANCE(M))
KEEPING SOURCE FUZZY SETS
CASE
    WHERE WITH .t.TweetID, .c.City, .c.Region
    GENERATE {
        TweetID: .t.TweetID,
        City: .c.City,
        Region: .c.Region
    }
DROP OTHERS
SET INTERMEDIATE .....
```

Esempio 5.1: SPATIAL JOIN con creazione di insieme fuzzy *distance_from_center*

```
GENERATE FUZZY OPERATOR distance_operator
PARAMETERS (distance TYPE FLOAT)
PRECONDITION distance >= 0
EVALUATE distance
RANGE (0, 1000)
POLYLINE (0, 1), (800, 1), (1000, 0)
```

Listato 5.1: Predicato fuzzy *distance_operator* per il calcolo della membership

Di seguito è riportato il risultato dell'applicazione dello SPATIAL JOIN definito nel Listato 5.2:

```
{
    "tweetID": 1,
    "City": "Bergamo",
    "Region": "Lombardia",
    "~geometry": {
```

```
        "coordinates": [9.69988419, 45.66594707],
        "type": "Point"
    },
    "~fuzzysets": {
        "distance_from_city_center": 0.7
    }
},
{
    "tweetID": 2,
    "City": "Bergamo",
    "Region": "Lombardia",
    "~geometry": {
        "coordinates": [9.69930355, 45.66561062],
        "type": "Point"
    },
    "~fuzzysets": {
        "distance_from_city_center": 0.7
    }
},
{
    "tweetID": 3,
    "City": "Bergamo",
    "Region": "Lombardia",
    "~geometry": {
        "coordinates": [9.69946617, 45.66563574],
        "type": "Point"
    },
    "~fuzzysets": {
        "distance_from_city_center": 0.2
    }
},
{
    "tweetID": 4,
    "City": "Bergamo",
    "Region": "Lombardia",
    "~geometry": {
        "coordinates": [9.69950141, 45.66532037],
```

```
        "type": "Point"
    },
    "~fuzzysets": {
        "distance_from_city_center": 0.6
    }
},
{
    "tweetID": 5,
    "City": "Bergamo",
    "Region": "Lombardia",
    "~geometry": {
        "coordinates": [9.69946387, 45.66533018],
        "type": "Point"
    },
    "~fuzzysets": {
        "distance_from_city_center": 0.6
    }
},
{
    "tweetID": 6,
    "City": "Bergamo",
    "Region": "Lombardia",
    "~geometry": {
        "coordinates": [9.69959869, 45.66584253],
        "type": "Point"
    },
    "~fuzzysets": {
        "distance_from_city_center": 0.8
    }
},
{
    "tweetID": 7,
    "City": "Bergamo",
    "Region": "Lombardia",
    "~geometry": {
        "coordinates": [9.69543276, 45.66445455],
        "type": "Point"
```

```
    },
    "~fuzzysets": {
        "distance_from_city_center": 1
    }
},
{
    "tweetID": 8,
    "City": "Bergamo",
    "Region": "Lombardia",
    "~geometry": {
        "coordinates": [9.6992909, 45.6657278],
        "type": "Point"
    },
    "~fuzzysets": {
        "distance_from_city_center": 0.8
    }
},
{
    "tweetID": 9,
    "City": "Bergamo",
    "Region": "Lombardia",
    "~geometry": {
        "coordinates": [9.69980096, 45.6633577],
        "type": "Point"
    },
    "~fuzzysets": {
        "distance_from_city_center": 1
    }
},
{
    "tweetID": 10,
    "City": "Bergamo",
    "Region": "Lombardia",
    "~geometry": {
        "coordinates": [9.69665857, 45.66715451],
        "type": "Point"
    },
```

```
    "~fuzzysets": {
        "distance_from_city_center": 1
    }
},
.....
```

Nella collezione risultante è stato quindi inserito il valore di membership relativo all'insieme fuzzy *distance_ from_ city_ center* grazie al predicato fuzzy DISTANCE valutato attraverso l'operatore *distance_ operator*.

Si veda ora un secondo esempio dove il calcolo della membership avviene direttamente attraverso la valutazione di un predicato fuzzy.

Si supponga di avere due collezioni. La prima che con nome *c1*:

```
{
    "id":"1001",
    "name": "rettangolo_1",
    "~geometry": {
        "type": "Polygon",
        "coordinates": [[45.70, 9.35],[45.70, 89.42],[45.67, 9.35],[45.67, 9.42]]
    }
},
.....
```

La seconda con nome *c2*:

```
{
    "id":"2002",
    "name": "rettangolo_2",
    "~geometry": {
        "type": "Polygon",
        "coordinates": [[45.73, 9.30],[45.73, 9.43],[45.68, 9.30],[45.68, 9.43]]
    }
},
.....
```

Si utilizza ora il predicato INSIDE(LEFT) sopra definito per il calcolo del valore di membership relativo all'inclusione delle due geometrie.

```
SPATIAL JOIN OF COLLECTIONS c1, c2
ON INTERSECT
SET GEOMETRY LEFT
SET FUZZY SETS inside_left
EVALUATE INSIDE(LEFT)
KEEPING SOURCE FUZZY SETS
CASE WHERE WITH c1.name, c1.id
    GENERATE {
        id: c1.id,
        name: c1.name
    }
DROP OTHERS
SET INTERMEDIATE AS .....
```

Esempio 5.2: Utilizzo del predicato fuzzy standard INSIDE per il calcolo della membership relativa all'insieme fuzzy *inside_left*.

Il risultato che si ottiene a seguito dell'esecuzione del codice proposto nell'Esempio 5.2 è il seguente:

```
{
    "id":"1001",
    "name": "rettangolo_1",
    "~geometry": {
        "type": "Polygon",
        "coordinates": [[45.70, 9.35],[45.70, 89.42],[45.67, 9.35],[45.67, 9.42]]
    },
    "~fuzzysets": {
        "inside_left": 0.32
    }
}
```

E' stato aggiunto l'insieme fuzzy *inside_left* il cui valore di membership è pari a 0.32.

Un ulteriore esempio, più complesso, potrebbe essere quello di verificare in che misura una determinata superficie abbia un perimetro la cui forma sia più o meno simile ad una circonferenza. Si supponga di avere la collezione *shape*.

Prima viene definito l'operatore *circular_shaped* e successivamente viene utilizzato all'interno della SPATIAL JOIN per verificare quanto la forma della geometria assomiglia ad una circonferenza.

```
CREATE FUZZY OPERATOR circular_shaped
PARAMETERS (area TYPE FLOAT, perimeter TYPE FLOAT)
PRECONDITIONS area > 0, perimeter > 0
EVALUATE (4 * 3.14 * area) / (perimeter * perimeter)
```

Predicato *circular_shaped*.

In questo caso i parametri passati all'operatore *circular_shaped* sono l'area e la distanza calcolati attraverso le rispettive funzioni.

```
SPATIAL JOIN OF COLLECTIONS shape, collection
ON INTERSECT
SET GEOMETRY LEFT
SET FUZZY SETS circular
EVALUATE circular_shaped(AREA(KM, .shape.~geometry),
                         PERIMETER(KM, .shape.~geometry))
KEEPING SOURCE FUZZY SETS
CASE WHERE .....
.....
```

Esempio 5.3: Utilizzo delle funzioni AREA e PERIMETER per il calcolo della membership relativa all'insieme fuzzy *circular*.

6 Sintassi Operatori Fuzzy

1. Create fuzzy operator

```
CREATE FUZZY OPERATOR operator_name
PARAMETERS (par_1 TYPE par_type, par_2 TYPE par_type, ...)
PRECONDITION conditions_on_parameters
EVALUATE expression
RANGE (xMIN, xMAX)
POLYLINE (x1, y1), (x2, y2), .... (xn, yn)
```

   RANGE e POLYLINE sono opzionali. Se non sono indicati assumono valore di default: RANG(0, 1) e POLYLINE(0,0), (1,1)

   L'*expression* può essere un semplice parametro oppure una qualunque espressione aritmetica. Inoltre può anche essere una funzione alla quale vengono passati tra parentesi tonde i parametri richiesti.

2. Generate fuzzy set

```
GET COLLECTION collection@db
FILTER
    CASE
        WHERE ...
        GENERATE FUZZY SET fuzzy_set_name
        EVALUATE fuzzy_expression
    DROP OTHERS
SET INTERMEDIATE ...
```

   fuzzy_expression: *expression* |
   IF_FAILS(operator_name(parameters), default_value_of_membership)
   Può essere presente più di un blocco GENERATE FUZZY SET ... EVALUATE ...

3. Query semplice con insiemi fuzzy

```
GET COLLECTION collection_name
FILTER
    CASE
        WHERE
```

```
            WITHIN FUZZY SETS name_of_fuzzy_set  //almeno un fuzzy_set
            ALPHA-CUT value_of_cut ON name_of_fuzzy_set
            DROPPING FUZZY SETS name_of_fuzzy_set, name_of_fuzzy_set, .....
DROP OTHERS
```

DROPPING FUZZY SETS è opzionale

4. Query complessa con insiemi fuzzy

```
GET COLLECTION collection_name@db
FILTER
    CASE
        WHERE
            KNOWN FUZZY SET fuzzy_set_name
            AND
            UNKNOWN FUZZY SET fuzzy_set_name
            GENERATE FUZZY SET fuzzy_set_name
            EVALUATE IF-FAILS(fuzzy_predicate(parameters),
                              membership_default_value)
        GENERATE FUZZY SET fuzzy_set_name
        EVALUATE expression
        ALPHA-CUT value_of_cut ON fuzzy_set_name
        DROPPING fuzzy_set_name, fuzzy_set_name, .....
KEEP OTHERS
.....
```

5. Javascript function

```
CREATE JAVASCRIPT FUNCTION function_name(parameters)
PARAMETERS (par_1 TYPE par_type, par_2 TYPE par_type, ...)
PRECONDITION conditions_on_parameters
BODY
{
    // istruzioni JavaScript
}
END BODY
```

6. Set fuzzy sets nel caso di JOIN standard:

```
JOIN OF COLLECTIONS collection_1, collection_2
SET FUZZY SETS collection_1.fuzzyset_1,
               collection_1.fuzzyset_2 AS fuzzyset_new_name,
               collection_2.fuzzyset_2, .....
DROPPING SOURCE FUZZY SETS | KEEPING SOURCE FUZZY SETS
```

7. Set fuzzy sets nel caso di SPATIAL JOIN:

```
SPATIAL JOIN OF COLLECTIONS C1, C2
ON .....
SET GEOMETRY .....
SET FUZZY SETS name_fuzzy_set
EVALUATE fuzzy_expression
.....
KEEPING SOURCE FUZZY SETS | DROPPING SOURCE FUZZY SETS
CASE
    WHERE WITH ....
```

fuzzy_expression: *standard_fuzzy_predicate* | *fuzzy_operator(standard_function)*

Indice