Prototipo Experimental para Consultas Difusas

Enrique González, Rosseline Rodríguez, Leonid Tineo

Universidad Simón Bolívar, Departamento de Computación
Caracas, Venezuela

RESUMEN

Los sistemas de consulta flexibles basados en lógica difusa mejoran la capacidad de resolver problemas complejos, en los gestores de base de datos. SQLf ha sido propuesto como un lenguaje que permite incluir elementos difusos en las instrucciones SQL. Sin embargo, resolver una consulta difusa añade un costo computacional al procesamiento tradicional de consultas. Se han propuesto cuatro estrategias diferentes de evaluación, para el procesamiento de consultas difusas, Este artículo presenta un prototipo diseñado para experimentar con estas estrategias a fin de proveer la capacidad de resolver consultas difusas y facilitar el estudio y comparación del comportamiento de los algoritmos basados en estas estrategias.

ABSTRACT

Flexible querying systems based on fuzzy logic enhance the capabilities of database systems to solve complex problems. SQLf has been proposed as a language that allows fuzzy elements to be included in SQL statements. However, solving fuzzy queries adds overhead to traditional query processing. Four different evaluation strategies have been proposed for fuzzy querying processing, This paper presents a prototype designed for experimentation with these strategies in order to both provide the ability to solve fuzzy queries and to study and compare the behavior of the algorithms based on these strategies.

Keywords: Fuzzy Database, Query Processing, SQLf.

1. Introducción

Las restricciones impuestas por la lógica booleana en los sistemas clásicos de base de datos, normalmente obligan a que problemas complejos sean simplificados o especializados más allá de lo originalmente deseado. Esto usualmente produce que soluciones válidas sean descartadas por el carácter restrictivo de los parámetros, o que soluciones no válidas sean incluidas por su laxitud. Además, no es trivial discriminar por su efectividad a un grupo de soluciones dadas, como respuesta a una consulta compleja.

Por otro lado, los sistemas de consultas difusas son capaces de manejar la noción de grados de verdad y pueden funcionar con un nivel cercano al lenguaje humano, usando expresiones tales como "alto" y "la mayoría". Además, en el modelo relacional difuso se provee un marco en el cual de forma automática cada solución a una consulta difusa contiene un valor que indica el grado en que ésta satisface la consulta especificada.

SQLf es una extensión difusa de SQL propuesta originalmente por Bosc y Pivert [BP95] y completada posteriormente por el grupo de Tineo [GGT09]. Este lenguaje de consulta permite el uso de condiciones difusas en cualquier lugar donde SQL permite una precisa. SQLf es el más completo pues incluye las características de SQL2003 [GGT09]. Existen también varios intérpretes y evaluadores SQLf [GT08].

Se han propuesto cuatro estrategias diferentes para la evaluación de consultas difusas [Tin06]. Estas estrategias se pueden utilizar alternativamente, cada una con un rendimiento variable en función de diferentes factores. Sin embargo, más investigación en el área es requerida, con el objetivo final de construir un sistema de consultas que puede elegir la mejor estrategia para evaluar una consulta difusa, de una manera similar a lo que un optimizador de bases de datos relacional lo haría en la actualidad.

Esto genera la necesidad de un sistema que pueda resolver toda la gama de consultas permitidas por SQLf, usando cualquiera de estas estrategias, con el fin de poder estudiar todas las posibles interacciones. Este sistema también debe proporcionar los medios para comparar de manera significativa el rendimiento de los algoritmos basados en dichas estrategias. Los sistemas de consultas SQLf existentes en la actualidad [GT08] [ACT11] no satisfacen estos requisitos, por lo que se decidió que la creación de un prototipo especializado para este fin sería más efectivo que la modificación de los sistemas previos.

Las cuatro estrategias de evaluación conocidas se basan en la traducción de la consulta difusa a una consulta clásica y el procesamiento del resultado mediante funciones o rutinas procedimentales que pueden implementarse en un manejador de bases de datos actual. Por esta razón se decidió realizar el prototipo con una arquitectura de acoplamiento intermedio [Tim01] en que el traductor es un pro-

grama externo al manejador, pero la funcionalidad añadida es codificada en el lenguaje nativo del manejador de bases de datos. Para el traductor se usó el lenguaje Prolog, ya que, por ser un lenguaje fundamentalmente declarativo y basado en estructuras, facilita el proceso de "parsing" y traducción. Oracle PL/SQL fue elegido como el lenguaje final en el cual se traduce una consulta para luego ser ejecutada. El prototipo, denominado SQLf-pl, debe su nombre a estos elementos.

Este trabajo está organizado como sigue: en la sección 2 se presentan los antecedentes de la investigación. La sección 3 muestra las capacidades del prototipo, detallando la sintaxis SQLf utilizada para especificar consultas difusas. La arquitectura del sistema se muestra en la sección 4. La sección 5 explica brevemente las estrategias propuestas para resolver consultas difusas. La sección 6 muestra las capacidades de experimentación que provee el prototipo. La sección 7 presenta las conclusiones y recomendaciones para trabajos futuros.

2. Antecedentes

La teoría de conjuntos difusos [Zad65] proporciona una base matemática para el modelado de conceptos tales como la incertidumbre y la imprecisión. La membresía de un elemento en un conjunto difuso puede ser cualquier valor en el intervalo [0,1] siendo 0 (falso) y 1 (verdadero). Esto permite que en la lógica difusa, cualquier predicado puede tener un valor de verdad en este mismo intervalo, en lugar de la dicotomía impuesta por la lógica de Boole.

Los términos vagos del lenguaje natural pueden ser representados y manipulados computacionalmente mediante conjuntos difusos. A los efectos de este trabajo, son de principal interés los términos vagos que permiten describir cantidades, los cuales se conocen como cuantificadores difusos. Por ejemplo, términos como "muchos", "pocos", "alrededor de 3", "la mayoría", "la mitad"- Es posible definir el cuantificador difuso "muchos", especificando el grado en que cada cantidad real se dice que representa a "muchos". Es decir, un cunatificador difuso se puede definir mediante un conjunto difuso.

Los cuantificadores difusos se dice que son absolutos cuando describen la cardinalidad del conjunto de elementos que satisfacen una condición. Estos cuantificadores se definem mediante conjuntos difusos en R^+. Un ejemplo de ellos es la expresión "alrededor de 3". Por otro lado, los cuantificadores difusos proporcionales describen la proporción de elementos de un conjunto base que satisfacen una condición. Éstos pueden ser definidos como conjuntos difusos en el intervalo real [0,1]. Un ejemplo de ellos sería la expresión "la mayoría"

Los cuantificadores también son clasificados en crecientes, decrecientes y unimodales. Los crecientes son aquellos cuya membresía aumenta a lo largo de la progresión de los números. En los decrecientes sucede lo contrario. Los unimodales, aumentan y disminuyen después. Es posible usar cuantificadores para formar expresiones lógicas. De especial interés son las fórmulas cuantificadas difusas de forma S1, que establecen que la cantidad de elementos de un conjunto clásico X que satisface el predicado difuso A es descrito por el cuantificador difuso Q. Estas instrucciones se representan como Q(X,A) o " Q X´s son A". También son relevantes las fórmulas cuantificadas de forma S3, las cuales establecen que la cantidad de predicados de una lista A_1, A_2 ... A_n que son satisfechas es descrita por el por el cuantificador Q. La expresión $Q(A_1,A_2...A_n)$ se utiliza para representar este tipo de instrucciones.

La lógica difusa ha sido aplicada en la implementación de varios sistemas para almacenar datos difusos y/o recuperar datos de acuerdo a una especificación difusa, tales como OMRON [NSA83], FQUERY [KZ95], F-SQL [GUP06], cada uno usando su propio lenguaje.

Como una propuesta para estandarizar la formulación de consultas difusas, Bosc y Pivert [BP95] definieron SQLf, una extensión difusa de SQL. SQLf es un lenguaje diseñado para realizar consultas difusas sobre bases de datos relacionales clásicas. Es decir, los datos almacenados no son difusos en sí mismo, sino que todos los elementos difusos del lenguaje son usados para determinar el grado en que una determinada consulta es satisfecha por los datos discretos existentes.

Esto permite que SQLf sea implementado como una capa sobre un manejador de bases de datos regular. Tal implementación ha sido desarrollada en la Universidad Simón Bolívar. Actualmente este software es conocido como SQLfi [GT08] e incluye las características del estándar SQL2003 [GGT09]. Fue implementado como una capa en Java sobre Oracle RDBMS y ha sido diseñado para ser usado en red por otras aplicaciones a través de un API remoto. Dicho prototipo utiliza lo que en este documento se conoce como la Estrategia de Programa Derivado. La ventaja principal provista por el prototipo SQLf-pl presentado en este trabajo sobre el sistema SQLfi es la capacidad de usar y comparar las distintas estrategias posibles para la resolución de consultas difusas.

3. Consultas en SQLf-pl

El prototipo SQLf-pl implementa una gran parte de la diversidad de consultas difusas en SQLf propuestas en trabajos anteriores [BP95][GT08][GGT09]. A continuación se presentan muy brevemente las características del lenguaje de consulta tal como lo soporta SQLf-pl.

Se proveer el lenguaje de definición de datos (DDL) de SQLf [Tin98] que permite la **definición de términos difusos** para su posterior uso. En SQLf-pl, los términos definidos no se almacenan persistentemente, pues su uso más allá del alcance de una sesión no se ha considerado necesario para los objetivos del sistema. La sintaxis de la definición varía según el tipo de término, pero en general tiene la siguiente estructura:

CREATE FUZZY *termType termName* AS *termDef*;

donde *termType* es una de las palabras reservadas PREDICATE, MODIFIER, COMPARATOR, CONNECTOR o QUANTIFIER que indica qué clase de término difuso es según [Zad77], *termName* es un identificador que designa al término y *termDef* una expresión compleja que define la semántica operacional del término según [BP95].

La sintaxis de una **consulta difusa básica** es muy similar a la de SQL, y tiene la estructura:

SELECT *A* FROM R_1, R_2, ..., R_n WHERE *fc*;

donde R_1, R_2, ..., R_n son tablas, *A* es una lista de atributos pertenecientes a dichas tablas y *fc* una condición difusa compuesta de términos difusos definidos.

Además del uso de condiciones difusas, una de las principales diferencias con SQL clásico es la inclusión del grado de satisfacción (μ) para cada fila del resultado.

Es posible especificar que las filas en el resultado deben satisfacer una cierta cantidad condiciones, describiendo esta cantidad con un cuantificador difuso, a eso se le conoce como **cuantificación difusa en bloque simple**. En este caso la sintaxis a ser usada es:

SELECT *A* FROM R_1, R_2, ..., R_n

WHERE Q(fc_1, fc_2,..., fc_m);

con Q un cuantificador difuso, fc_1, fc_2,..., fc_m las condiciones difusas y el resto como en la definición previa. Note que Q(fc_1, fc_2,..., fc_m) es una forma S3.

Se permite el **anidamiento de bloques de consulta al estilo del operador EXISTS** en SQL. En este caso el operador EXISTS puede ser sustituido por cualquier cuantificador difuso absoluto, resultado en la siguiente estructura:

SELECT *A* FROM R_1, R_2, ..., R_n

WHERE Qa

(SELECT *B* FROM T_1, T_2,..., T_m WHERE *fc*);

con Qa un cuantificador difuso absoluto, T_1, T_2,..., T_m tablas, *B* una lista de atributos que pertenecen a dichas tablas y el resto como en las definiciones previas.

El otro tipo de anidamiento provisto permite comparar un valor de un ciclo externo con uno de un ciclo interno, esto es **anidamiento al estilo de los operadores ANY y ALL** de SQL. Estos operadores son sustituidos por cualquier cuantificador difuso

SELECT *A* FROM R_1, R_2, ..., R_n

WHERE A_i Q $\prec f$

(SELECT B_1 FROM T_1, T_2,..., T_m WHERE *fc*);

con Q un cuantificador difuso absoluto, $\prec f$ un comparador difuso, A_i un atributo perteneciente a *A*, B_1 un atributo perteneciente a alguna tabla de T_1, T_2,..., T_m, y el resto como en las definiciones previas. Además, Q debe ser un cuantificador absoluto o *fc* debe ser una condición booleanas, de manera que la instrucción pueda interpretarse como una forma S1.

También se da soporte a **consultas particionadas con cuantificadores difusos** en SQLf con la sintaxis:

SELECT *A* FROM R_1, R_2, ..., R_n

WHERE *bc* GROUP BY *A'* HAVING Q ARE *fc*;

con *A'* un subconjunto de los atributos listados en *A* y el resto como en las definiciones previas. Es posible tener una expresión de la forma Q fc_1 ARE fc_2 en la cláusula HAVING si Q es un cuantificador absoluto o si fc_1 es una condición booleanas, ya que en estos casos la expresión puede ser reducida a una forma simple como se explica en la sección 5.5.

La **cláusula FOR SOME (o FOR ALL)** puede ser usada en consultas anidadas de SQLf [GT08] para cuantificar los elementos de un ciclo interno que deban satisfacer alguna condición difusa. Este trabajo propone la generalización de la estructura para que provea el uso de cualquier cuantificador difuso en lugar de sólo ser limitado a cuantificación existencial y universal, usando la siguiente sintaxis

SELECT *A* FROM R_1, R_2, ..., R_n

WHERE Q (Ψ) ARE *fc*;

con Ψ una sub-consulta difusa, y el resto como en las definiciones previas. El prototipo SQLf-pl actualmente puede resolver este tipo de consultas si Q es un cuantificador absoluto, pero no para cuantificadores relativos.

Similarmente la **cláusula UNIQUE** puede ser usada en un anidamiento para determinar de manera difusa si hay duplicados en el resultado de una sub-consulta

SELECT *A* FROM R_1, R_2, ..., R_n

WHERE UNIQUE (Ψ);

con la misma nomenclatura que en las definiciones previas. El prototipo SQLf-pl es el primero en resolver este tipo de consultas. Sin embargo, éste sólo puede usar la estrategia ingenua para este caso. El algoritmo usado para resolver estas consultas fue propuesto en [Gon06].

Las consultas pueden usar **vistas difusas** definidas previamente en la cláusula FROM. Las tuplas que pertenecen a las vistas tiene un grado de satisfacción en el resultado al menos igual al que ellas satisfacen en la definición de la vista. Las vistas difusas también pueden ser definidas anónimamente dentro de una consulta, especificando la definición (una sub-consulta) dentro de paréntesis en la cláusula FROM en lugar del nombre de una tabla.

Todos los tipos anteriores de consultas se pueden combinar usando los operadores de conjuntos o de join en la misma manera que en el SQL clásico. Las mismas restricciones aplican, como compatibilidad de atributos para el conjunto de operaciones. Aunque la sintaxis es la misma, estos operadores son interpretados de acuerdo al Algebra Relacional Difusa [UF94]. Los operadores sobre resultados que actualmente acepta el prototipo SQLf-pl son UNION, INTERSECT, EXCEPT, JOIN, CROSS JOIN, NATURAL JOIN y OUTER JOIN.

Para todas las consultas, **la cláusula WITH CALIBRATION** puede ser añadida al final de la consulta para especificar el grado mínimo para el cual una tupla que satisface la consulta debe aparecer en el resultado

SELECT ... WITH CALIBRATION λ;

donde $\lambda \in [0,1]$. Sólo un grado de calibración es permitido por consulta. Cuando las sub-consultas o los operadores sobre resultados son usados, todas las consultas involucradas son calibradas usando el mismo valor. Aunque la calibración es opcional, todas las estrategias menos la ingenua dependen de la existencia de un valor calibración. En la figura 1, se muestra un ejemplo de una consulta difusa y su correspondiente respuesta, arrojada por el prototipo SQL-

pl. Cada tupla respuesta tiene asociado el grado de membresía (μ) al conjunto resultado.

```
SQLf>
SELECT name, age FROM student s1
 WHERE some
 ( SELECT age from student
   WHERE s1.age doubles age )
 WITH CALIBRATION 0.3;

MU          NAME         AGE
1.0         Locke         67
0.730769    Teri          45
0.576923    Michael       41
0.44        Derek         36
0.44        Nina          36
0.375       Sawyer        34
0.333333    Nazi          32
0.333333    Jack          32
-- 8 rows selected --
```

Figura 1: *Consulta y Resultado en SQLf-pl*

4. Arquitectura del Sistema

SQLf-pl se implementó usando SWI-Prolog sobre el ORACLE 9i. Para conectar estos dos subsistemas, se usó un pequeño componente JAVA, tomando ventaja de los protocolos de comunicación JPL y JDBC.

La comunicación directa SWI Prolog-ORACLE fue considerada, pero finalmente fue descartada debido a la falta de disponibilidad de herramientas eficaces y accesibles para esa tarea. El subsistema de Prolog se compone de diversos archivos de código fuente agrupados en directorios de acuerdo a su función. A pesar que SWI-Prolog ofrece la opción de forzar estrictamente la modularización, éste fue descartado pues la división del código Prolog en módulos hacía más difícil el uso de las herramientas de prueba que fueron parte del proceso de desarrollo. Los "módulos" se aplicaron sólo para la estructura de directorios, los estándares de codificación y la documentación.

Dentro del código de Prolog, se siguió una organización de tres capas, lo cual se muestra en la figura 2. Dos módulos manejan la entrada y salida usuario formando la capa de interfaz, mientras que un solo módulo maneja la interacción con la base de datos. En la capa lógica, un módulo está dedicado exclusivamente a analizar de los comandos y consultas, otro se encarga de las funciones relacionadas con el análisis de desempeño, y el resto ejecuta cada una de las instrucciones en alguno de los sub-lenguajes SQLf. Cabe señalar que el módulo de DML, que se encarga de la traducción de la consulta, es bastante más complejo que los módulos del DAL y del DDL. Esta organización ofrece, entre otras facilidades, un pipeline para la resolución de la consulta.

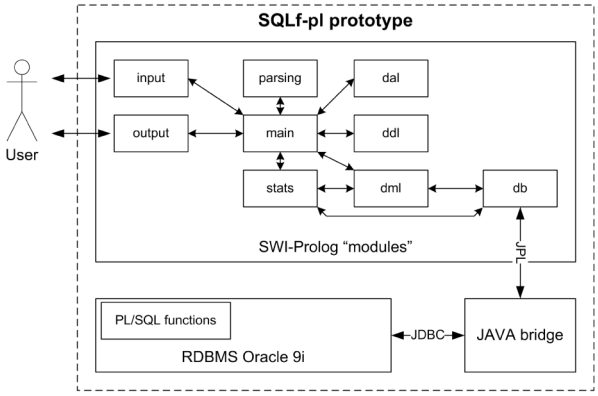

Figura 2: *Arquitectura del prototipo SQLf-pl*

Una consulta difusa primero va al módulo de parsing, la cual es una gramática de cláusulas definidas que convierte la consulta en un árbol sintáctico. Esta representación del árbol permite una manipulación fácil de la consulta dentro de Prolog. Cuando el árbol de la consulta es recibido por el módulo del DML pueden aplicarse transformaciones en caso de ser necesario.

Las consultas que usan cunatifiadores difusos decrecientes o unimodales son transformadas en consultas equivalentes que sólo usan cuantificadores crecientes, para ello se implementan transformaciones definidas y probadas previamente [Tin06]. Las consultas que involucran vistas y operadores sobre resultados son también transformados para reducir su complejidad, aunque esto resulta en la necesidad de realizar varias consultas sencillas en lugar de una más compleja.

Las transformaciones permiten dar soporte a toda la variedad de consultas del lenguaje, reduciendo el número de casos particulares de traducciones para la evaluación de las consultas en el manejador relacional.

La traducción de la consulta difusa resulta en otro árbol sintáctico que representa el programa o la consulta clásica que se pasará al manejador. Después de un post-procesamiento, el árbol se representa como una cadena de código por el módulo de base de datos.

El manejador Oracle 9i recibe una consulta SQL o un bloque de código PL/SQL, que son inicialmente analizables y comprensibles por éste. Sin embargo, ambos hacen uso de funciones que calculan el grado de satisfacción de cada fila, lo cual debe ser suministrada por el manejador dentro del conjunto resultado. Estas funciones deben estar disponibles en los archivos de código PL/SQL que serán cargados por Oracle antes de la puesta en marcha del prototipo SQLf-pl.

5. Estrategias de Evaluación

En trabajos anteriores [BP95][BLP00][Tin06] se han propuesto cuatro estrategias para traducir una consulta difusa en un programa o consulta clásica, que pueda ejecutarse con un manejador relacional, a fin de dar respuesta a

la consulta difusa. El prototipo SQLf-pl implementa estas cuatro estrategias, las cuales se describen a continuación.

La **Estrategia de Programa Ingenuo** resuelve una consulta recuperando todas las filas que potencialmente pueden aparecer en el resultado, usando una versión simplificada de la consulta que elimina las condiciones difusas. Luego, un programa externo calcula el grado de satisfacción de cada fila y descarta aquellas que no tienen el valor requerido. Dado que esto tiende a acceder muchas filas de datos que son innecesarios, esta estrategia suele ser la menos eficiente, sin embargo, es también la más simple.

La **Estrategia de Programa Sugeno** se basa en la interpretación de fórmulas cuantificadas difusas como una integral difusa de Sugeno [BLP00]. La fórmula de la integral puede ser usada en una consulta anidada o particionada para obtener el número mínimo de filas del ciclo interno que debe satisfacer la condición difusa con el fin de obtener el grado de satisfacción requerido en el ciclo externo. Los algoritmos basados en esta estrategia pueden descartar un grupo de filas cuando sea evidente que el mínimo no se alcanzará, por lo que el resto del grupo nunca se accederá.

La **Estrategia de Programa Derivado** aplica el principio de la derivación a una consulta difusa para obtener la consulta derivada. Ésta es la consulta clásica recuperará las filas que satisfacen la consulta difusa original con al menos el grado de satisfacción requerido. Como el resultado debe indicar el grado de satisfacción de cada fila, se usa un programa que calcula este valor para cada elemento en el resultado de la consulta derivada. Esta estrategia es aplicable en todos los casos cubiertos por el principio de derivación.

La **Estrategia de Consulta Derivada** también utiliza el principio de derivación, pero el grado de satisfacción de cada fila se calcula usando una función dentro de la consulta derivada. El manejador realiza este cálculo al mismo tiempo que resuelve la consulta, lo que en teoría permite una optimización. Obviamente, esta estrategia no se puede utilizar si el manejador no acepta llamadas a funciones dentro de una consulta.

6. Capacidades para Experimentación

El prototipo SQLf-pl puede obtener y registrar una serie de estadísticas relacionadas con los recursos aplicados para resolver cualquier consulta en particular. Para acceder a esta funcionalidad un simple lenguaje de administración de datos ha sido provisto.

Las estadísticas que se registran en el subsistema de Prolog son: el tiempo total consumido, el tiempo de utilización de CPU, memoria heap y stack usada, y las inferencias lógicas necesitadas. Éstas se registran en un archivo cuya estructura puede observarse en la figura 3.

El análisis sólo puede ser hecho sobre el proceso de traducción (desde el parsing del de la consulta al envío del programa traducido al DBMS), o los recursos usados en buscar, procesar y mostrar los datos deben también ser tomados en cuenta. El nombre del archivo de texto en el cual registrar las estadísticas se especifica con la instrucción:

SET STATS TRANSLATION *filePathName*

```
Total Time Spent (sec) ..................... 13
CPU Time Spent (msec) ...................... 70
Logic Inferences ........................ 11045
Memory Heap Used (bytes) ................ 4688
Memory Stack Size Used (bytes) ......... 17684
Total Memory Used (bytes) ............... 22388
```

Figura 3: *Archivo de estadísticas de SQLf-pl*

Por otro lado, es posible obtener el desempeño de las estadísticas en cuanto a los recursos empleados por el DBMS para ejecutar la consulta traducida. Estas estadísticas son provistas por lo general por el DBMS Oracle: las operaciones parser, execute y fetch; el tiempo total y el tiempo de CPU, lecturas físicas y lógicas, registros procesados y fallos de memoria caché. Una porción de archivo de estadísticas Oracle que pueden ser obtenidas a través de SQL-pl se muestra en la figura 4. Análogo el caso anterior, se provee el comando:

SET STATS DBMS *filePathName*

```
/opt/oracle/admin/ora9i/udump/ora9i_ora_1242
4_sqlfpl_1153086104_27.trc

*** TRACE DUMP CONTINUED FROM FILE  ***

Oracle9i Enterprise Edition Release
9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data
Mining options
JServer Release 9.2.0.1.0 - Production
ORACLE_HOME = /opt/oracle/product/9.2.0
```

Figura 4: *Porción de archivo de estadisticas de Oracle obtenido con SQLf-pl*

Por último, algunas funciones se añadieron al prototipo SQLf-pl para contribuir a la tarea de realizar experimentos elaborados para estudiar y comparar el rendimiento de las diferentes estrategias que resuelven consultas difusas.

La función más obvia es la capacidad de seleccionar la estrategia a ser utilizada por el prototipo para resolver las consultas, tal como se muestra en la figura 5.

```
SQLf>
SET STRATEGY SUGENO_PROGRAM;

Strategy set to sugeno_program

SQLf>
SELECT s1.name, s2.name
 FROM student s1, student s2
 where s1.bloodtype donates s2.bloodtype
 with calibration 0.7;

WARNING: strategy sugeno_program is invalid.
Strategy naive_program was used.
```

Figura 5: *Selección de estrategia en SQLf-pl*

Si por alguna razón el prototipo no puede usar la estrategia elegida por el usuario, un mensaje de advertencia se presenta y se toma la estretagia ingenua (como es el caso de la Figura 5.). Para indicar el nombre de la estrategia deseada (SUGENO_PROGRAM, DERIVED_PROGRAM, DERIVED_QUERY, NAIVE) se provee la instrucción:

SET STRATEGY *strategyName*

El usuario experto probablemente estará interesado en conocer exactamente qué comandos están siendo ejecutados por el manejador para procesar una consulta difusa. Por lo tanto, se proporciona la opción de ver el código que resulta de la traducción. Este código es automáticamente indentado para que sea más legible. El nombre del archivo de comandos en PL/SQL en el cual generar el código se especifica con la instrucción:

SET TRANSLATION *filePathName*

Extracto del código que genera SQL-pl para la estrategia Consulta Derivada puede observarse en la figura 6, y para la estrategia Programa Derivado en la figura 7.

```
SELECT
    to_char(
        mu_s1_abs_inc(
            mu_query('(
                SELECT
                    mu_trapezoid( . . . )
                FROM
                    student
                WHERE
                    (:sqlfpl#s1$#$age  . . .
            )',
            str_table('sqlfpl#s1$#$age'),
            str_table(s1.age)),
            0,
            8
        ),
        'S0D000000'
    ) AS mu,
    name,
    age
FROM
    student s1
WHERE (
    1
    ) <= (
    SELECT
        count(*)
    FROM
        student
    WHERE
        (s1.age / age) >= 1.1
    )
ORDER BY
    mu DESC
```

Figura 6: *Extracto de código generado por SQLf-pl para estrategia Consulta Derivada*

También es posible diseñar consultas y experimentos de antemano y luego ejecutarlas en modo batch. Cualquier comando que puede ser ejecutado en la cónsola interactiva también puede incluirse en un archivo batch, permitiendo la ejecución de archivos de manera jerárquica. Si una consulta particular presenta problemas, se pasará a la siguiente, a fin de permitir tanto como sea posible que el proceso batch sea finalizado. La carga del archivo de comandos se hace mediante la intrucción:

LOAD *filePathName*

Junto con esta funcionalidad se tiene la capacidad de escribir toda la salida pantalla a un archivo de texto, lo cual puede cambiarse siempre que sea necesario, así como los datos de salida pueden organizarse como se desee. Para esto se proveen los comandos:

SET OUTPUT *filePathName*

SET OUTPUT STANDARD

```
BEGIN
    FOR top_tuple IN top_cursor LOOP
        current_group := str_table();
        current_group.EXTEND(1);
        sup_outer := 0;
        FOR tuple IN sub_cursor(top_tuple
            current_group(1) := 'sqlfpl';
            IF not(tuple.bloodtype = curr
                current_group(1) := tuple
                sup_g :=
                    mu_trapezoid(tuple.age,
                sup2_g := 0;
```

Figura 7: *Extracto de código generado por SQLf-pl para la estrategia Programa Ingenuo*

7. Conclusiones

Este documento ilustra el desarrollo de un prototipo que permite la resolución de consultas difusas especificadas en SQLf y provee los medios para analizar el desempeño de las diferentes estrategias que pueden ser usadas para resolver dichas consultas. El prototipo permite el uso de predicados, modificadores, conectores, comparadores, cuantificadores y vistas difusas; así como también los operadores de anidamiento EXISTS-like, ANY-like, FOR Q y UNIQUE; particionamiento, operadores resultado, entre otras estructuras. La mayoría de éstos pueden ser resueltos con la posibilidad de elegir entre cuatro estrategias: Programa Ingenuo, Programa Sugeno, Programa Derivado y Consulta Derivada. Además, este artículo propone la extensión de SQLf para generalizar anidamiento FOR SOME / FOR ALL en el anidamiento FOR Q, lo cual permite el uso de cualquier cuantificador difuso. Este prototipo es asimismo la primera aplicación conocida del anidamiento ÚNIQUE difuso. La finalización con éxito de este prototipo muestra la validez de los algoritmos propuestos previamente [Tin06] como implementaciones de las estrategias para resolver consultas difusas. También se muestra que las transformaciones pueden ser usadas para reducir el tamaño y la complejidad de un sistema de consultas difusas. Varias vías de estudios futuros pueden explorarse. Un algoritmo para resolver anidamiento FOR Q usando un cuantificador proporcional todavía no se ha diseñado. Sin embargo, es posible que una generalización de los algoritmos usados para el anidamiento ANY-like pueda resolver este problema. Aunque el SQLf-pl fue diseñado para su uso con pocos algoritmos de traducción, confiando en las transformaciones, sería interesante determinar si el enfoque opuesto tiene alguna ventaja. Finalmente, es claro que la especificación actual de SQLf no tiene el mismo poder expresivo que SQL, ya que sólo permite dos niveles de anidamiento. La posibilidad de permitir múltiples anidamientos difusos debe ser estudiada en el futuro.

Referencias

[ACT11] Aguilera, A., Cadenas, J., Tineo, L., "Fuzzy Querying Capability at Core of a RDBMS", En: "Advanced database query systems : techniques, applications and Technologies" Li Yan and Zongmin Ma, editors. pp. 160-184, 2011.

[BLP00] Bosc, P., Liétard, L. & Pivert, O., "Evaluation of Flexible Queries: The Quantified Statement Case", Proceedings of the 8th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems IPMU'2000, Madrid, España.

[BP95] Bosc P. & Pivert O. "SQLf: A Relational Database Language for Fuzzy Querying", IEEE Transactions on Fuzzy Systems, Vol 3, No. 1, Feb 1995.

[Cod70] Codd, E. F. "A Relational Model of Data for Large Shared Data Bank", Communication of ACM, Vol 13. No. 6. 1970

[GUP06] Galindo J., Urrutia, A. Piattini, M., Fuzzy Database Modeling, Design and Implementation. Idea Group Publishing. 2006.

[Gon06] González, E. "Sistema Experimental de Consultas Difusas a Bases de Datos SQLf-pl". Informe final de Proyecto de Grado, Universidad Simón Bolívar, 2006.

[GGT09] González, C., Goncalves, M., Tineo, L. A New Upgrade to SQLf: Towards a Standard in Fuzzy Databases. En: 20th International Workshop on Database and Expert Systems Application Proceeding, pp. 442-446. 2009.

[GT08] Goncalves, M., Tineo, L. SQLfi y Sus Aplicaciones. En: Revista Avances en Sistemas e Informática, Vol. 5(2), pp. 33-40. 2008.

[KZ95] J.Kacprzyk & S.Zadrozny, "Fuzzy Queries in Microsoft AccessTM v.2", Proceedings of FuzzyIEEE'95 Workshop on Fuzzy Database Systems and Information Retrieval, pp 61-66, 1995.

[NSA83] H.Nakajima, T.Sogoh, M.Arao "Fuzzy Database Language and Library-FuzzyExtension to SQL", Proceedings of Second IEEE International Conference on Fuzzy Systems, pp 477-482, 1983.

[Tim01] Timarán, R. "Arquitecturas de integración del proceso de descubrimiento de conocimiento con sistemas de gestión de bases de datos: Un estado del arte. Universidad del Valle, Colombia. Ingeniería y Competitividad, 3(2), 44–51. 2001.

[Tin00] Tineo, L. "Extending RDBMS for allowing Fuzzy Quantified Queries", Proceedings of DEXA 2000.

[Tin98] Tineo, L. "Interrogaciones Flexibles en Bases de Datos Relacionales". Universidad Simón Bolívar, Venezuela, 1998. Trabajo de Ascenso presentado ante la Universidad Simón Bolívar.

[Tin06] Tineo, L. "Una contribución a la interrogación flexible de bases de datos: evaluación de consultas cuanticadas difusas". Universidad Simón Bolívar, Venezuela, 2006. Tesis Doctoral.

[UF94] Umano, M. & Fukami, S. "Fuzzy Relational Algebra for Possibility-Distribution-Fuzzy-Relational Model of Fuzzy Data", Journal of Intelligent Information System, Vol 3. pp 7-27. 1994.

[Zad65] Zadeh, L. "Fuzzy Sets". Information and Control 8, 1965.

[Zad77] Zadeh, L. A. "PRUF – A Language for the Representation of Meaning in Natural Languages.". IJCAI 918.1977