INGENIERIA EN INFORMÁTICA
Memoria del Proyecto Final de Carrera

NQL (Natural Query Language)

Autor: Daniel Boerner

Consultor: Jordi Duran Cals

Enero - 2013

Resumen

En paralelo al aumento de los datos que se crean en nuestro entorno actual y que se procesan mediante gestores de bases de datos, crecen las necesidades de información de usuarios de sistemas de información. Como usuarios, no necesariamente disponen de los conocimientos técnicos necesarios para extraer la información que requieren. Así pues, conviene simplificar y facilitar el acceso a la información almacenada en sistemas de bases de datos.

En este sentido y enmarcado como proyecto final de carrera de los estudios de ingeniería informática y específicamente del área de compiladores, presento el desarrollo de un sistema capaz de procesar consultas en lenguaje natural introducidas por el usuario mediante el teclado. El sistema es capaz de responder a consultas en castellano, relacionadas con un dominio de aplicación representado mediante una base de datos relacional.

A través de la presente memoria, primeramente introduzco el proyecto mediante su motivación, sus objetivos, su planificación, sus resultados y los requisitos del sistema presentado. Después expongo el principal problema relacionado con el procesamiento del lenguaje natural: la ambigüedad. Sigue una breve descripción del lenguaje estructurado de consultas *SQL* (*Structured Query Language*) concentrada en la sentencia *SELECT*, utilizada para recuperar determinadas filas de una tabla de una base de datos. Como punto central, describo el diseño y la implementación de un sistema interfaz entre un lenguaje natural y una base de datos relacional. A continuación describo las limitaciones del sistema desarrollado así como las pruebas realizadas sobre el mismo. Finalmente y después de una conclusión y un apartado de referencias siguen como anexos, detalles sobre la instalación y ejecución del sistema, las preguntas que reconoce el sistema y finalmente las especificaciones del léxico y de la gramática correspondientes a la implementación del núcleo del sistema.

Contenido

Figuras

1. Introducción

Cada día es más importante obtener información a partir de los datos almacenados en bases de datos. La cantidad de usuarios que requieren información aumenta con el tiempo y las consultas de información son cada vez más complejas. Todo ello conlleva a una necesidad de simplificar y optimizar la interacción entre los usuarios y los sistemas de información que procesan bases de datos.

En este sentido, un primer paso fue en su día el desarrollo, la normalización, la implementación y la aceptación del lenguaje SQL. Aún conociendo este lenguaje y disponiendo de cierta habilidad en su uso, para obtener la información deseada es necesario un conocimiento de la estructura de la base de datos relacional que se consulta. Esto presenta un importante obstáculo a usuarios que no disponen de dichos conocimientos.

1.1. Análisis del problema

Desde finales de los años sesenta se investiga en sistemas *NLIDB* (*Natural Language Interfaces to Databases*) que permiten al usuario acceder a la información almacenada en una base de datos tecleando consultas expresadas en un lenguaje natural.

Aunque algunos de los sistemas desarrollados a mediados de los años ochenta presentan impresionantes características aplicados a áreas determinadas, su aceptación y comercialización no ha progresado mucho desde entonces.

En un principio, los usuarios de estos sistemas no necesitan aprender un lenguaje artificial como SQL para interactuar con la base de datos. En la práctica, sin embargo, los sistemas en cuestión suelen responder únicamente a un subconjunto del lenguaje natural limitado por el área de aplicación del sistema. Cuando el sistema no es capaz de responder a una determinada consulta, resulta difícil para el usuario determinar si el motivo es la limitada capacidad lingüística del sistema o conceptualmente su consulta se sale fuera del ámbito de aplicación del mismo. Por ello es necesario que el usuario aprenda los limites y las capacidades del sistema lo cual puede presentar una dificultad de aprendizaje comparable al aprendizaje de un lenguaje artificial.

Otros posibles inconvenientes que impiden la proliferación de estos sistemas se derivan usualmente de su compleja configuración y difícil portabilidad entre diferentes áreas de aplicación o bases de datos.

Pero el principal motivo de la poca aceptación y comercialización de dichos sistemas son los problemas relacionados con el procesamiento del lenguaje natural y más concretamente con la ambigüedad intrínseca que presentan los lenguajes naturales.

A pesar de importantes avances, el procesamiento informático del lenguaje natural aún se encuentra en una fase inicial de desarrollo. Aún así, si se limita el dominio de aplicación es posible reducir las dificultades que presenta el procesamiento informático del lenguaje natural. De esta forma, utilizando simples técnicas o herramientas para el análisis léxico y sintáctico se puede conseguir cierta efectividad en la interacción mediante un lenguaje natural con sistemas de procesamiento de base de datos.

1.2. Objetivos

Los objetivos del proyecto que describe esta memoria son cuatro:

1. El principal objetivo se centra en el desarrollo e implementación de un interfaz en lenguaje natural entre usuarios sin conocimiento de SQL y un sistema de gestión de bases de datos relacionales.

El sistema, denominado *NQL*, traduce preguntas realizadas por el usuario en lenguaje natural a consultas SQL. Estas consultas SQL son ejecutadas por un gestor de bases de datos y el sistema muestra al usuario los resultados de dicha ejecución.

A efectos de mostrar el diseño y las funcionalidades de NQL, el dominio de aplicación se limita a un calendario de eventos representado por una base de datos relacional. Dada una tabla de personas, una tabla de eventos y una tabla de asistentes, NQL es capaz de responder a preguntas, como por ejemplo:

- *Tengo alguna reunión mañana por la mañana?*
- *Cuantas personas han confirmado la asistencia a la reunión de hoy?*
- *Cuantos días faltan para la próxima reunión con Pedro?*
- *Cuantas reuniones sobre presupuestos realizaremos este mes?*

2. Un segundo objetivo es analizar de forma introductoria los inconvenientes que debe afrontar un sistema NLIDB. Algunos de ellos ya los he mencionado en el apartado introductorio de este documento. Las dificultades derivadas del problema de la ambigüedad de los lenguajes naturales las expongo en un apartado propio.

3. Como tercer objetivo, pretendo introducir el lenguaje artificial SQL concentrándome en la parte que utilizo para la implementación del sistema NQL.

4. Finalmente, el último objetivo es aprovechar y reforzar los conocimientos que he adquirido en las asignaturas *Compiladors I*, *Compiladors II*, *Metodologia i gestió de projectes informàtics*.

1.3. Estructura de la memoria

Esta memoria consta de las siguientes secciones.

Sección 1. Introducción

En esta sección presento un análisis del problema que este proyecto intenta solventar. También enumero los objetivos de este proyecto.

Sección 2. Estudio de viabilidad

En esta sección expongo los recursos temporales y humanos dedicados al proyecto así como las partes interesadas en el mismo. Después de detallar los resultados esperados del proyecto, sigue una breve exposición del procesamiento del lenguaje natural y las dificultades que presenta la ambigüedad. Después de una corta descripción de SQL y la sentencia SELECT, detallo los requisitos del sistema NQL y describo tres sistemas que he tomado como referencia para desarrollar el sistema NQL.

Sección 3. Diseño e implementación de NQL

Esta sección incluye la arquitectura diseñada y el entorno de desarrollo elegido para desarrollar el sistema NQL. Después de describir los principales componentes que componen el sistema, explico el tratamiento de errores que realiza NQL.

Sección 4. Limitaciones y pruebas del sistema

Esta sección enumera algunas de las limitaciones más importantes que presenta el sistema NQL y en ella describo también las pruebas que he realizado para comprobar el funcionamiento del sistema.

Sección 5. Conclusiones

En esta sección expongo posibles ampliaciones que se pueden implementar sobre el sistema NQL, valoraciones personales sobre la realización del proyecto y termino la sección con una conclusión final.

Sección 6. Referencias

En esta sección listo todo el material que he utilizado como fuente de información para la realización de este proyecto.

Sección 7. Anexos

Esta sección incluye un anexo sobre la instalación y ejecución del sistema NQL, un anexo que enumera todas las consultas reconocidas por NQL, un anexo con las especificaciones léxicas del núcleo de NQL y finalmente, un anexo con las especificaciones sintácticas del núcleo de NQL.

2. Estudio de viabilidad

En esta sección expongo la planificación temporal realizada para desarrollar el proyecto. A continuación detallo los resultados obtenidos de la ejecución del proyecto. En otro apartado relaciono las partes interesadas en el proyecto así como los recursos humanos dedicados al mismo. Sigo con una breve exposición al procesamiento en lenguaje natural, detallando las dificultades que presenta la ambigüedad. Después de una breve introducción a SQL y la sentencia SELECT, expongo los requisitos del sistema NQL. Finalmente finalizo esta sección con una presentación de tres sistemas ya existentes que he tomado como referencia para el desarrollo de NQL.

2.1. Planificación

He planificado el proyecto teniendo en cuenta como principales hitos, las fechas de entrega correspondientes a las siguientes pruebas de evaluación continua (PAC):

- PAC 1: 03/10/2012
- PAC 2: 07/11/2012
- PAC 3: 12/12/2012

A continuación presento la planificación detallada de cada fase del proyecto, especificando las actividades a realizar y los productos a obtener por cada fecha o periodo:

Fecha	19/09/2012
Actividades	• Inicio del proyecto

	• Realización de la portada y de la plantilla para la memoria del proyecto
Productos	• Portada y plantilla para la memoria del proyecto

Periodo	19/09/2012 – 03/10/2012
Actividades	• Estudio del procesamiento del lenguaje natural y su aplicación en la interacción con bases de datos • Estudio de sistemas existentes • Confección del plan de trabajo • Entrega PAC1
Productos	• Plan de trabajo

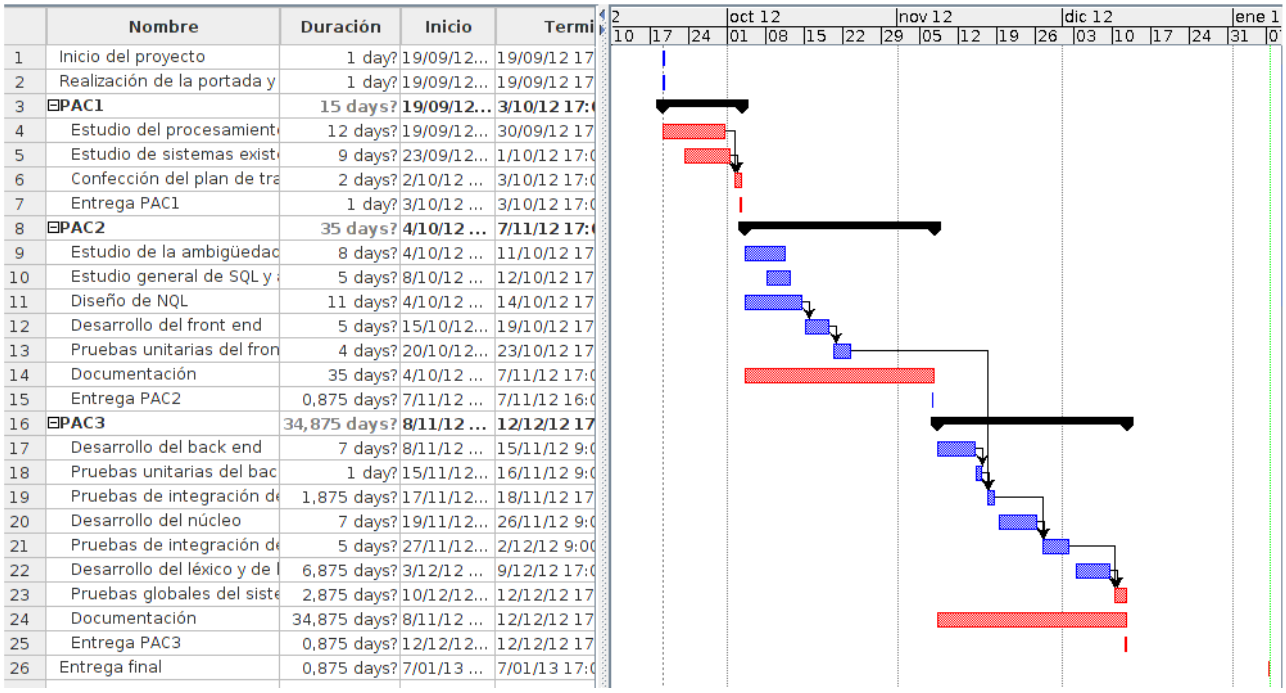
Periodo	04/10/2012 – 07/11/2012
Actividades	• Estudio de la ambigüedad en el procesamiento del lenguaje natural y su impacto en la interacción con bases de datos • Estudio general de SQL y análisis de las posibilidades de la sentencia SELECT • Diseño de NQL • Desarrollo del front end • Pruebas unitarias del front end • Documentación • Entrega PAC2
Productos	• Implementación del front end de NQL • Documento de diseño de NQL

Periodo	08/11/2012 – 12/12/2012
Actividades	• Desarrollo del back end • Pruebas unitarias del back end • Pruebas de integración del front end y el back end • Desarrollo del núcleo • Pruebas de integración del núcleo, front end y el back end

	• Desarrollo del léxico y de la gramática del núcleo • Pruebas globales del sistema NQL • Documentación • Entrega PAC3
Productos	• Implementación del back end de NQL • Implementación del núcleo de NQL • Implementación de NQL • Documento de la implementación de NQL

Fecha	07/01/2013
Actividades	• Entrega final
Productos	• Memoria del proyecto • Sistema NQL

A continuación muestro el diagrama temporal asociado a la planificación del proyecto.



	Nombre	Duración	Inicio	Termi
1	Inicio del proyecto	1 day?	19/09/12...	19/09/12 17
2	Realización de la portada y	1 day?	19/09/12...	19/09/12 17
3	**PAC1**	**15 days?**	**19/09/12...**	**3/10/12 17:**
4	Estudio del procesamient	12 days?	19/09/12...	30/09/12 17
5	Estudio de sistemas exist	9 days?	23/09/12...	1/10/12 17:0
6	Confección del plan de tra	2 days?	2/10/12 ...	3/10/12 17:0
7	Entrega PAC1	1 day?	3/10/12 ...	3/10/12 17:0
8	**PAC2**	**35 days?**	**4/10/12 ...**	**7/11/12 17:**
9	Estudio de la ambigüedac	8 days?	4/10/12 ...	11/10/12 17
10	Estudio general de SQL y	5 days?	8/10/12 ...	12/10/12 17
11	Diseño de NQL	11 days?	4/10/12 ...	14/10/12 17
12	Desarrollo del front end	5 days?	15/10/12...	19/10/12 17
13	Pruebas unitarias del fron	4 days?	20/10/12...	23/10/12 17
14	Documentación	35 days?	4/10/12 ...	7/11/12 17:0
15	Entrega PAC2	0,875 days?	7/11/12 ...	7/11/12 16:0
16	**PAC3**	**34,875 days?**	**8/11/12 ...**	**12/12/12 17**
17	Desarrollo del back end	7 days?	8/11/12 ...	15/11/12 9:0
18	Pruebas unitarias del bac	1 day?	15/11/12...	16/11/12 9:0
19	Pruebas de integración d	1,875 days?	17/11/12...	18/11/12 17
20	Desarrollo del núcleo	7 days?	19/11/12...	26/11/12 9:0
21	Pruebas de integración d	5 days?	27/11/12...	2/12/12 9:00
22	Desarrollo del léxico y de l	6,875 days?	3/12/12 ...	9/12/12 17:0
23	Pruebas globales del siste	2,875 days?	10/12/12...	12/12/12 17
24	Documentación	34,875 days?	8/11/12 ...	12/12/12 17
25	Entrega PAC3	0,875 days?	12/12/12...	12/12/12 17
26	Entrega final	0,875 days?	7/01/13 ...	7/01/13 17:0

Figura 1. Diagrama temporal de la planificación del proyecto

En base a los días del periodo de realización del proyecto y la dedicación media diaria, calculo un esfuerzo de 300 horas para la realización del proyecto.

2.2. Resultados

Como resultados o productos de este proyecto final de carrera se tienen la presente memoria como documento descriptivo del trabajo realizado para la obtención del sistema NLIDB presentado así como el propio sistema NQL y una presentación en video del proyecto.

2.3. Partes interesadas

Como proyecto final de carrera, hay dos partes interesadas en el trabajo y los resultados que presenta esta memoria. Primeramente, el autor de la misma como estudiante e interesado principal y después el personal docente responsable de la supervisión y evaluación del proyecto.

2.4. Recursos humanos

Como proyecto final de carrera, los resultados del mismo han sido realizados de forma individual por el autor de esta memoria.

2.5. Presupuesto

Dado que las herramientas utilizadas para la realización de este proyecto son de libre distribución y sin costes de licencia, el presupuesto de este proyecto se limita a la cotización de las horas de trabajo dedicadas a la realización del proyecto y especificadas al final del apartado 2.1, dedicado a la planificación.

2.6. Riesgos

A continuación enumero los riesgos que contemplo para este proyecto:

- No encontrar las herramientas adecuadas
  - Probabilidad: media
  - Contingencia: considerar herramientas alternativas
- Baja por enfermedad común

- Probabilidad: media
- Contingencia: recortar funcionalidades de los requisitos.

2.7. El procesamiento del lenguaje natural

El procesamiento del lenguaje natural es un área de la ciencia de la computación, la inteligencia artificial y la lingüística, enfocada a la interacción entre ordenadores y lenguajes humanos (naturales).

Entre sus principales objetivos se encuentra el diseño y desarrollo de programas que analizan, entienden y generan lenguajes que, de forma natural es utilizado por los seres humanos.

Conseguir este objetivo no es nada fácil. Entender un lenguaje natural significa entre otras cosas, saber que conceptos representa una palabra o frase y saber como relacionar estos conceptos entre sí para determinar cierto significado.

2.7.1. La ambigüedad en el procesamiento del lenguaje natural

Los lenguajes naturales se caracterizan por su ambigüedad y redundancia. Ello facilita a los seres humanos el aprendizaje del lenguaje pero, actualmente dificulta enormemente su procesamiento automático mediante ordenadores.

La ambigüedad es una característica inherente a los lenguajes naturales y las personas la perciben de diferentes formas en base a sus particulares esquemas de interpretación. Por ejemplo, personas que intentan comunicarse en un lenguaje natural que están aprendiendo son propensas a interpretar palabras de una forma excesivamente literal o a usar palabras fuera de contexto aunque su significado sea el correcto.

Básicamente, los seres humanos resuelven la ambigüedad a través del contexto en el cual se enmarca su discurso, conversación o mensaje. Cuando se trata de lenguaje hablado, la ambigüedad puede ser resuelta incluso por la entonación de las palabras o frases.

En el procesamiento del lenguaje natural, se define comúnmente la ambigüedad como "*la posibilidad de ser interpretado de varias formas diferentes*". Se puede clasificar la ambigüedad de diversas maneras aunque la más usual es aquella que la divide en

ambigüedad *léxica*, en la cual una palabra puede tener más de un significado, en ambigüedad *semántica*, la cual contempla las diferentes formas en que se pueden combinar los significados de varias palabras que forman parte de una misma frase, en ambigüedad *sintáctica*, la que contempla las diferentes formas de estructurar gramaticalmente una secuencia de palabras y en ambigüedad *pragmática*, la que considera las diferentes interpretaciones de una frase en función de su contexto.

Se conocen además otros conceptos lingüísticos asociados a la ambigüedad que pueden conllevar a interpretaciones erróneas. La *generalización*, en la que se utiliza una palabra cuyo significado engloba el significado más especifico de otra palabra, la *indeterminación*, en la que existen elementos qué, aunque no ayuden a entender un significado son necesarios para concretar a qué o quien se refieren y la *vaguedad* que da lugar a cierta incertidumbre sobre un significado.

Los sistemas que contemplan la posibilidad de interacción conversacional deben afrontar además las *anáforas* y las frases *elípticas*. Se trata de construcciones sintácticas que siguen siendo gramaticalmente correctas a pesar de eliminar alguna palabra del discurso. En estos casos, para una interpretación correcta se precisa referenciar algún antecedente o estado conversacional anterior en el tiempo.

A continuación muestro algunos ejemplos de ambigüedad.

Lista todos los empleados de la empresa con carnet de conducir.

El sistema debe decidir si el determinante (carnet de conducir) se aplica sobre los empleados o sobre la empresa. Para ello debería conocer que una empresa no puede tener carnet de conducir.

Lista todos los usuarios de Sevilla y de Cádiz.

El sistema debe saber que un usuario solo puede ser de Sevilla o de Cádiz.

Cuales son las mayores empresas?

El sistema debe saber que métrica utilizar para determinar las mayores empresas, por ejemplo, volumen de ventas o cantidad de empleados.

Quien obtuvo más ingresos?

El sistema debe determinar a quien se refiere la pregunta.

El perro me trae el periódico cada mañana?

El sistema no puede saber si cada mañana se trata del mismo periódico o cada mañana es un periódico diferente.

En general, para resolver al máximo la ambigüedad, un sistema de procesamiento del lenguaje natural necesita procesar una enorme cantidad de información relacionada principalmente con el dominio de aplicación.

Con el objetivo de superar las dificultades inducidas por la ambigüedad, el diseño de NQL implementa una relación directa entre las frases que introduce el usuario y las que es capaz de procesar el sistema. De esta forma, NQL dispone de la máxima información para descartar cualquier interpretación errónea.

2.8. La sentencia SELECT de SQL

El lenguaje estructurado de consultas SQL es un lenguaje de programación diseñado específicamente para gestionar datos mediante un sistema gestor de bases de datos relacionales.

SQL fue uno de los primeros lenguajes comerciales basados en el modelo relacional ideado por Edgar F. Codd en el año 1970. Aunque contempla elementos procedurales, SQL es considerado un lenguaje declarativo, también denominado de cuarta generación (4GL). En el año 1986, el *ANSI* (*American National Standards Institute*) estandarizó SQL. Lo mismo hizo la *ISO* (*International Organization for Standards*) en el año 1987. Desde entonces, se ha revisado varias veces el estándar con el fin de adaptarlo y mejorarlo con nuevas funcionalidades. La revisión del estándar del año 1992, denominada SQL-92, es la versión más extendida y aceptada habiendo sido adoptada e implementada por la mayoría de fabricantes de sistemas de gestión de bases de datos.

A pesar de existir un estándar, muchos fabricantes de sistemas de gestión de bases de datos incorporan extensiones y funcionalidades propias ampliando de forma particular las posibilidades de SQL pero creando problemas de portabilidad de las aplicaciones desarrolladas en SQL.

El lenguaje SQL se puede descomponer en tres partes principales:

- Lenguaje de definición de datos
- Lenguaje de control de datos
- Lenguaje de manipulación de datos

Mediante el *lenguaje de definición de datos* se gestionan las tablas y sus estructuras. Las sentencias más comúnmente utilizadas son CREATE, ALTER, RENAME, DROP y TRUNCATE:

- CREATE crea un objeto (tabla, índice, etc...) de la base de datos.
- ALTER modifica la estructura de un objeto de la base de datos.
- RENAME cambia el nombre de un objeto de la base de datos.
- DROP elimina un objeto de la base de datos.
- TRUNCATE elimina los datos contenidos en una tabla de la base de datos.

Mediante el *lenguaje de control de datos* se autoriza a los usuarios el acceso y manipulación de los datos. Las principales sentencias de esta parte son GRANT y REVOKE:

- GRANT autoriza a uno o varios usuarios a realizar un conjunto de operaciones sobre un objeto (tabla, índice, etc...) de la base de datos.
- REVOKE elimina autorizaciones realizadas mediante la sentencia GRANT.

Mediante el *lenguaje de manipulación de datos* se añaden, modifican, recuperan y eliminan datos. autoriza a los usuarios el acceso y manipulación de los datos. Las sentencias más comúnmente utilizadas son INSERT, UPDATE, SELECT y DELETE:

- INSERT añade nuevas filas a una tabla.
- UPDATE modifica un conjunto de filas de una tabla.
- SELECT recupera un conjunto de filas de una o varias tablas.
- DELETE elimina un conjunto de filas de una tabla.

Los operadores básicos de SQL son:

Operador	Descripción
=	Igual
<>	Diferente
>	Mayor
<	Menor
>=	Mayor o igual
<=	Menor o igual
BETWEEN	Entre un rango inclusive

LIKE	Según un patrón
IN	En un conjunto de valores

Además de estos operadores, SQL dispone también de funciones, como por ejemplo:

Función	Descripción
COUNT	Cuenta el número de filas
MIN	Calcula el mínimo de una columna determinada
MAX	Calcula el máximo de una columna determinada
SUM	Suma los valores de una columna determinada
AVG	Calcula la media de una columna determinada
NOW	Devuelve la fecha y el tiempo actual
DATE	Crea y devuelve una fecha como un objeto en una representación interna

En cuanto a tipos de datos, SQL contempla los siguientes tipos básicos:

- Cadena de caracteres de longitud fija o variable (y máxima *n*):
  - CHAR(*n*)
  - VARCHAR(*n*)
- Cadena de bits de longitud fija o variable (y máxima *n*):
  - BIT(*n*)
  - BIT VARYING(*n*)
- Número
  - INTEGER y SMALLINT
  - FLOAT, REAL y DOUBLE PRECISION
- Fecha y tiempo
  - DATE
  - TIME
  - TIMESTAMP

Una sentencia SELECT recupera cero o más filas de una o varias tablas de una base de datos. SELECT es la acción más compleja de SQL y en la mayoría de aplicaciones es la operación más utilizada. Por formar parte de un lenguaje declarativo, una operación SELECT especifica un conjunto de filas como resultado de una consulta pero no

especifica como obtener dicho conjunto. El gestor de la base de datos transforma la consulta en un plan de consulta que difiere entre diferentes ejecuciones, entre diferentes versiones de bases de datos y entre diferentes sistemas de gestión de bases de datos. Esta transformación es realizada típicamente por un componente denominado optimizador de consultas. La funcionalidad de este componente es calcular, bajo determinadas condiciones, la mejor ejecución posible de la consulta.

Desde el punto de vista sintáctico, una consulta incluye después de la palabra reservada SELECT, una lista de columnas que serán incluidas en el conjunto resultante de su ejecución. De modo comodín, se puede sustituir la lista de columnas por un símbolo asterisco para indicar que se requieren todas las columnas de las tablas referenciadas. Opcionalmente y siguiendo el orden mostrado, se pueden especificar además las siguientes palabras reservadas:

- FROM indica la tabla o las tablas de las cuales se quieren obtener datos.
- WHERE especifica una condición que limita las filas obtenidas por la consulta. Se eliminan todas aquellas filas del resultado para las cuales la condición no es evaluada como verdadera.
- GROUP BY agrupa las filas obtenidas en grupos en base a alguna función especificada.
- HAVING en base a una función especificada, filtra las filas resultantes de una agrupación obtenida mediante GROUP BY.
- ORDER BY ordena de forma ascendente o descendente el conjunto de filas obtenidas en base a las columnas especificadas.

De forma compacta y utilizando una notación típica para especificar gramáticas, muestro a continuación la sintaxis de una sentencia SELECT:

```
SELECT [ALL | DISTINCT] [ * | [columna [, columna …]]]
        [FROM tablas
                [WHERE condición]
                [GROUP BY {columna}
                [HAVING condición]
                [ORDER BY {columna} [ASC | DESC]]]
```

Mediante unos ejemplos de consultas generadas por NQL describo otros detalles de una sentencia SELECT.

```
select Apellido1 as 'Primer apellido' from Persona where
DNI='12345678Z'
```

En este ejemplo muestro una consulta SELECT en su estructura más sencilla. Se consulta la tabla Persona para recuperar el primer apellido de una persona cuyo DNI se corresponde al especificado en la condición. Como aspecto adicional, se puede ver la palabra reservada *as*. Mediante la misma es posible nombrar la cabecera de la columna de resultados de la consulta. En este caso particular, en lugar de denominarse igual que el nombre de la columna en la base de datos ("Apellido1"), la columna resultante se mostrará con una cabecera denominada "Primer apellido". Con ello pretendo facilitar al usuario la interpretación los valores contenidos en la columna recuperada.

```
select (case count(*) when 0 then 'no' else 'si' end) as 'Respuesta'
from Persona where Nombre='Juan' and Apellido1='Pastor'
```

En este ejemplo muestro una consulta SELECT algo más compleja. En función de la existencia de una persona cuyo nombre sea "Juan" y su primer apellido sea "Pastor", el resultado de la consulta será literalmente "si" o "no". Para ello utilizo una función para contar el número de filas que cumplen la condición. Una vez contado el número de personas que cumplen con la condición, condicionalmente al resultado de la función *count y* mediante el elemento *case…when...then…else…,* la consulta SELECT devuelve como resultado una u otra cadena de caracteres. Al igual que el ejemplo anterior, con ello pretendo facilitar al usuario la interpretación de los resultados.

```
select min(Hora_inicio) as 'Hora Inicio' from Evento where Fecha =
(select min(Fecha) from Evento where Tema='Impuestos' and Fecha >=
Date(Now()))
```

En este último ejemplo muestro una consulta SELECT que considera en una de sus condiciones el resultado de otra consulta SELECT. La segunda consulta devuelve la primera fecha de las próximas reuniones sobre impuestos. De esta forma, la primera

consulta devuelve la primera hora de inicio de la próxima reunión sobre impuestos. Así pues, SQL permite encajar consultas SELECT en otras consultas SELECT.

2.9. Requisitos del sistema

Dado un dominio de aplicación representado mediante una base de datos relacional, se trata de procesar en tiempo real consultas en un determinado lenguaje natural introducidas por un usuario y relacionadas con el tema de un determinado dominio de aplicación.

En un principio, tanto el dominio de aplicación como el lenguaje natural seleccionado deben estar dentro del alcance del proyecto. El dominio de aplicación no debe ser demasiado complejo para poder así satisfacer los recursos temporales y humanos disponibles según la planificación expuesta al inicio de esta memoria. Por otro lado, el lenguaje natural elegido debe ser entendido y aceptado en todos los niveles por todas las partes interesadas en este proyecto.

La introducción de consultas se realiza a través de un interfaz gráfico de usuario. Cada consulta está formada por una única frase de longitud indeterminada. La interacción entre el usuario y el sistema es del tipo consulta-respuesta. No se considera una interacción conversacional.

Se debe suponer que las consultas introducidas por el usuario están correctamente formadas desde un punto de vista ortográfico y gramático. De esta forma no se espera ninguna corrección automática por parte del sistema (ver apartado 5.5.1 *Correcciones y sugerencias*).

El sistema debe procesar consultas en lenguaje natural en forma de primera persona. Esto quiere decir, que el sistema debe ser capaz de determinar que usuario está interactuando con él.

Como casos de uso, se tiene por un lado, al usuario que interactua con el sistema realizando consultas en lenguaje natural y por otro lado, se tiene un administrador del sistema que lo instala y configura.

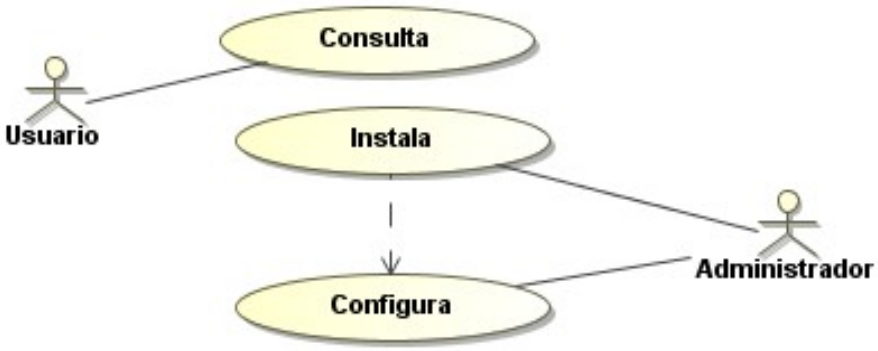


Figura 2. Diagrama de casos de uso

No se contempla la necesidad de introducción de datos ya que se parte de una base de datos con contenido. De esta forma y aunque SQL contemple sentencias de actualización, creación y eliminación de objetos de una base de datos, únicamente se considera la realización de consultas sobre objetos de la base de datos.

El procesamiento de la consulta en lenguaje natural se centra en su traducción a una única consulta SQL para poder ser ejecutada por el sistema gestor de bases de datos relacionales. Cada una de las consultas en lenguaje natural reconocidas determina por lo tanto la consulta SQL generada por el sistema.

Una vez generada una consulta SQL, el sistema la ejecuta a través del gestor de base de datos y la prepara los resultados devueltos por este para su visualización por parte del usuario.

Así pues, el interfaz gráfico debe mostrar los resultados de la consulta SQL generada y ejecutada. Además muestra una evolución de las diferentes acciones que realiza el sistema durante su inicialización, durante el procesamiento de cada consulta en lenguaje natural y durante la generación y ejecución de la correspondiente consulta SQL. Como acciones a notificar se contemplan también situaciones de error o conjuntos vacíos como resultado de una consulta.

En cuanto a la integración de la base de datos, el sistema debe disponer de una forma básica para configurar los datos de conexión con el gestor de base de datos. El objetivo es facilitar la portabilidad del sistema entre diferentes gestores de bases de datos.

Conviene facilitar al máximo la adaptación del sistema a otros dominios de aplicación y permitir la ampliación del conjunto de posibles consultas en lenguaje natural. Para ello, el sistema debe registrar aquellas consultas introducidas por el usuario que el sistema no fue capaz de responder. De esta forma, el administrador del sistema las analizará y en su caso reconfigurará el sistema para que el sistema acepte dichas consultas.

Finalmente y como requisitos no funcionales, al tratarse de un sistema interactivo se requiere un tiempo de respuesta razonable.
Como entorno de ejecución y desarrollo, se consideran un punto de trabajo estándar de la UOC y software de código abierto y sin costes monetarios de licencias o distribución. No se requiere ningún lenguaje de programación, ningún sistema gestor de base de datos y ningún entorno de desarrollo en particular.

2.10. Sistemas de referencia

Como referencia y punto de partida he tomado tres sistemas, *NLBean*, *NLP-Reduce* y *SQ-HAL*, cuyo código fuente esta disponible públicamente.

En al año 1997, Mark Watson publicó un sistema NLIDB experimental desarrollado en Java y lo denominó NLBean. Durante su inicialización, este sistema analiza la estructura de la base de datos para extraer el nombre de las tablas y columnas. Una vez inicializado, el sistema está preparado para procesar consultas en inglés como lenguaje natural. La estructura de las frases reconocidas se corresponde directamente con la estructura de la consulta SELECT en su forma más simple. Así, reconoce frases formadas con los nombres de los objetos de la base de datos, o sinónimos de estos, y combinados mediante conjunciones (and/or) y condiciones (where). Ofrece muchas posibilidades de configuración a través de un interfaz gráfico de usuario, permitiendo incluso añadir tablas a la base de datos. Dada la sencillez de la estructura de las frases reconocidas, el procesamiento de las consultas en lenguaje natural es muy primitivo. Para ello no utiliza herramientas para el análisis léxico y sintáctico.

Aunque no se trata de un NLIDB de forma estricta, NLP-Reduce presenta algunas ideas que se pueden trasladar a un NLIDB. Esta desarrollado en Java y fue publicado en el año 2007 por Esther Kaufmann, Abraham Bernstein, and Lorenz Fischer de la Universidad de Zürich. NLP-Reduce está pensado para facilitar el acceso a la web semántica. En lugar de acceder a bases de datos relacionales, accede a ontologías asociadas a la web semántica. De esta forma genera consultas en un lenguaje

específico denominado SPARQL. Procesa consultas en inglés como lenguaje natural y para su análisis léxico y sintáctico tampoco utiliza herramientas clásicas de construcción de compiladores. De forma automática y a partir de una ontología relacionada con el dominio de aplicación genera un léxico. Mediante una base de datos léxica (WordNet) enriquece el léxico añadiendo los posibles sinónimos de cada palabra. También añade a cada palabra del léxico su lema reduciéndola a su raíz (stemming). Procesa la consulta en lenguaje natural descomponiéndola en tripletes de palabras. Mediante estos tripletes indexa el léxico enriquecido para determinar cierto significado y poder así generar la correspondiente consulta en SPARQL.

SQ-HAL fue publicado en el año 2000 por Supun Ruwanpura de la Universidad de Monash. Este sistema NLIDB esta desarrollado en Perl y a diferencia de los dos anteriores utiliza, para procesar consultas en inglés como lenguaje natural una herramienta de análisis léxico y sintáctico específica para la construcción de compiladores. Este sistema también realiza un análisis de la base de datos para conocer los objetos que la forman. Mediante un interfaz gráfico de usuario, permite también la configuración manual de sinónimos de los objetos de la base de datos. Con esta información y en base a una determinada gramática y acciones asociadas a cada una de sus producciones, un analizador léxico y sintáctico genera la correspondiente consulta SELECT.

El sistema que presento en este proyecto esta influenciado por algunas ideas de los tres sistemas anteriormente descritos. Concretamente el back end esta influenciado por NLBean, el front end por NLP-Reduce y el núcleo por SQ-HAL.

3. Diseño e implementación de NQL

En esta sección la arquitectura ideada para desarrollar el sistema NQL. Una vez determinada la arquitectura del sistema, describo el entorno de desarrollo seleccionado. En este apartado expongo la estructura utilizada para compilar todos los componentes que forman parte de la arquitectura del sistema. A continuación expongo los detalles de implementación de los tres principales componentes del sistema: el interfaz gráfico de usuario, el núcleo y la base de datos. Finalizo esta sección describiendo el tratamiento de errores que realiza el sistema NQL.

3.1. Arquitectura del sistema

Siguiendo una estructura modular he diseñado el sistema descomponiéndolo en varios componentes principales.

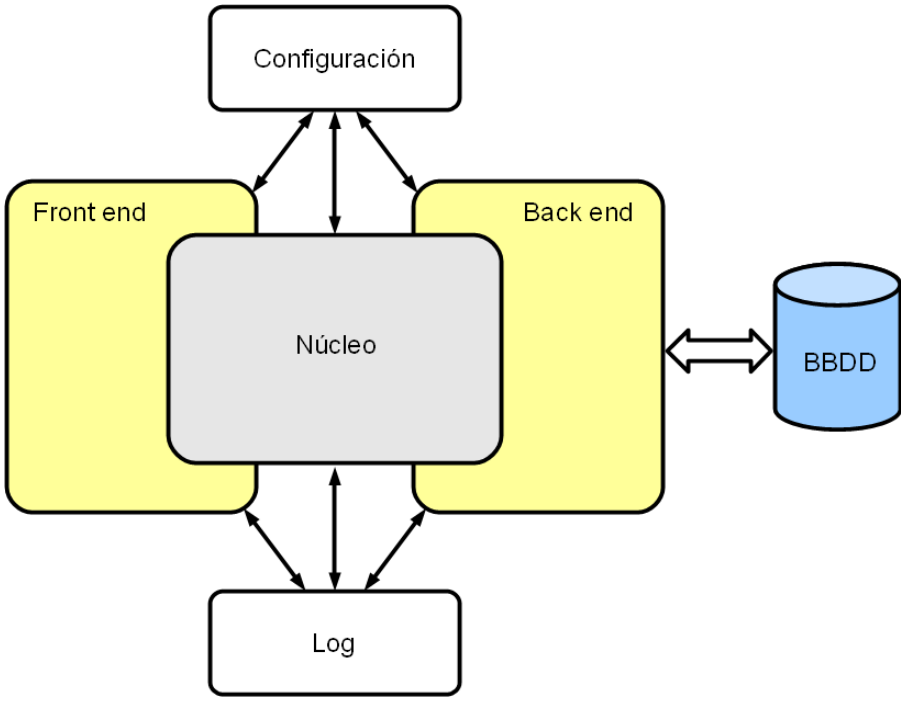


Figura 3. Arquitectura del sistema NQL

El front end consiste básicamente en el interfaz gráfico de usuario cuyas funcionalidades son las siguientes.

- *Introducción de consultas*

  En un elemento de texto de una línea, el usuario introducirá la consulta en lenguaje natural.

- *Invocación del procesamiento de consultas*

  Mediante un botón se inicia el procesamiento de la consulta introducida por el usuario y se muestran los resultados de dicho proceso.

- *Visualización de las consultas SQL generadas*

  La consulta SQL generada por el sistema se visualiza en un elemento de texto de solo lectura.

- *Visualización de resultados*

  De forma tabular se muestran las filas devueltas por la ejecución de la consulta SQL.

- *Visualización de acciones*

  Mediante mensajes informativos, se describen cada una de las acciones que el sistema realiza durante el procesamiento de cada consulta introducida por el usuario.

Mediante ficheros de configuración se pueden configurar el identificador del usuario y los parámetros del log y de la conexión del sistema con la base de datos.

Con el objetivo de poder corregir o ampliar el conjunto de consultas en lenguaje natural que el sistema es capaz de procesar, conviene tener constancia de aquellas consultas introducidas por el usuario que el sistema no ha sido capaz de traducir a una consulta SQL. A tal fin, el sistema dispone de un archivo log en el qué además, se registrarán todas las acciones realizadas por el sistema y los mensajes de error generados por el mismo.

El back end encapsula la base de datos del dominio de aplicación. A través de él se ejecutarán las consultas SQL generadas por el núcleo y también se procesaran los resultados obtenidos de las mismas para su visualización a través del front end.

El núcleo es el componente más interesante del sistema. Está formado por un analizador léxico y un analizador sintáctico integrados entre sí. En base a unas especificaciones léxicas y sintácticas se procesa la composición y estructura de las consultas en lenguaje natural reconocidas por el sistema. Como resultado de este proceso se genera para cada consulta en lenguaje natural la correspondiente consulta SQL.

En el siguiente diagrama de secuencia muestro la interacción entre los principales componentes del sistema.

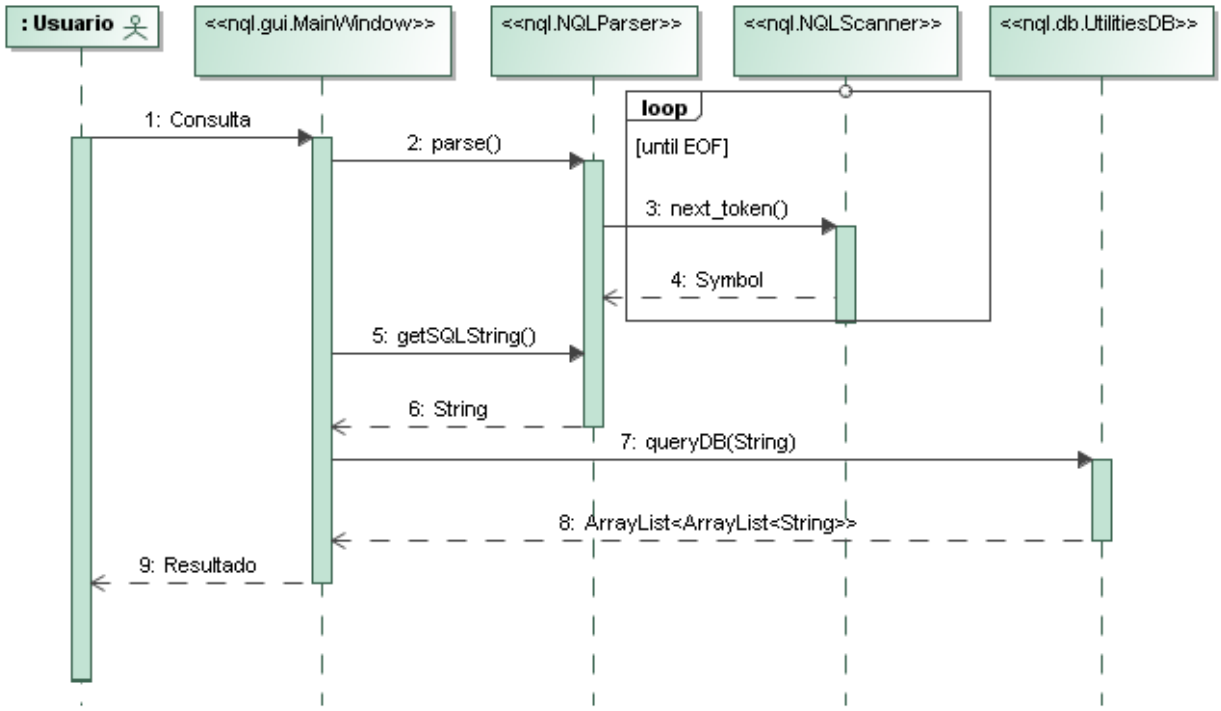


Figura 4. Diagrama de secuencia del sistema NQL

Una vez que el front end captura la consulta introducida por el usuario, se invoca al analizador sintáctico que forma parte del núcleo. Este está integrado con un analizador léxico que da soporte al análisis sintáctico de la consulta del usuario. Una vez finalizado el análisis léxico y sintáctico, el front end obtiene del núcleo la consulta SQL generada. A continuación el front end invoca al back end enviándole dicha consulta SQL para que este la someta al gestor de la base de datos para su ejecución. El back end obtiene el resultado de dicha ejecución y lo prepara para enviarlo al front end. Una vez recibidos por el front end, los resultados se visualizan a través del interfaz gráfico al usuario.

3.2. Entorno de desarrollo

Dados los conocimientos adquiridos al de cursar las asignaturas *Compiladors I* y *Compiladors II* en el manejo de las herramientas empleadas en dichas asignaturas he optado por utilizar las mismas para implementar el sistema NQL. Como generador de analizadores léxicos utilizo *JLex* y como generador de analizadores sintácticos utilizo *CUP*. En consecuencia he desarrollado el sistema NQL en el lenguaje de programación Java y sobre el entorno de desarrollo Eclipse. Para permitir, de forma alternativa, la compilación del sistema NQL sin la necesidad de *Eclipse*, he incorporado a la estructura de desarrollo un archivo de especificación para la herramienta de compilación

automática *Ant*. En dicho archivo incluyo además tareas que permiten interactuar con la base de datos para crear tablas, eliminar tablas e insertar datos en las tablas. Estas tareas pueden resultar útiles durante la fase de desarrollo y de pruebas.

Para el desarrollo de la base de datos que implementa el dominio de aplicación utilizo el gestor de bases de datos *MySQL*. Además de ser popular, es fácil de usar y conseguir.

Perl es una alternativa al lenguaje de programación Java utilizado para este proyecto. Se trata de un lenguaje muy potente para el procesamiento de texto. Para ello dispone además de una gran cantidad de librerías como por ejemplo, un generador de analizadores léxico y sintácticos, denominado Parse::RecDescent.
Sin embargo, al valorarse para este proyecto final de carrera, la utilización de las herramientas utilizadas durante las asignaturas *Compiladors I* y *Compiladors II* y para evitar complicaciones en la instalación y ejecución del interfaz gráfico del sistema NQL sobre la plataforma *Windows*, he optado finalmente por el entorno centrado en Java.

A continuación muestro la estructura de directorios y archivos que he utilizado en Eclipse para el desarrollo del sistema NQL.

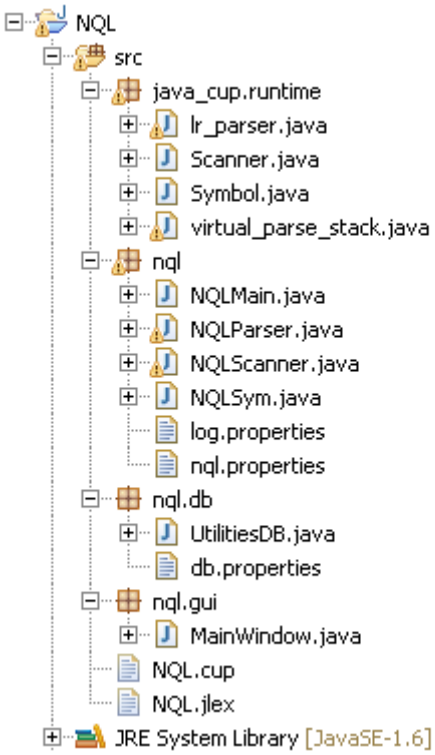


Figura 5. Estructura de desarrollo en Eclipse

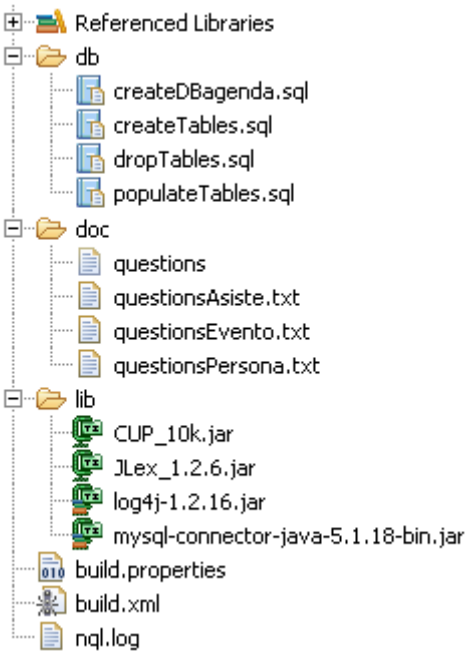


Figura 5. Estructura de desarrollo en Eclipse (continuación)

El primer componente del directorio *src* contiene el código fuente de las clases necesarias para la compilación del analizador sintáctico generado por CUP. Al ser CUP de código abierto, estas se obtienen de la misma distribución de CUP.

El siguiente componente contiene el núcleo del sistema. Por un lado se tiene la clase *NQLMain.java* que contiene el método principal para la ejecución del sistema NQL. Además de ejecutar el sistema y a partir del archivo de configuración *nql.properties*, el método principal obtiene el identificador del usuario para el cual ha sido configurado el sistema. En esta clase también se configuran las propiedades de la ventana del interfaz de usuario y se crea una instancia del mismo.
La clase *NQLParser.java* contiene el código del analizador sintáctico generado por CUP y la clase *NQLScanner.java* contiene el código del analizador léxico generado por JLex. La clase *NQLSym.java* es generada por CUP y contiene el código de los símbolos utilizados por el analizador sintáctico. Estas tres clases se generan mediante la herramienta Ant con las especificaciones de compilación automática contenidas en el archivo *build.xml* y a partir de las especificaciones contenidas en los archivos *NQL.jlex* y *NQL.cup*.
El archivo de configuración *log.properties* especifica entre otras configuraciones de la funcionalidad log, el archivo destino para registrar las anotaciones, el formato de las anotaciones, el nivel de anotaciones, etc... El formato de este archivo es muy simple y

sigue las convenciones de Java: propiedad=valor.

En el archivo de configuración *nql.properties* se almacena el identificador que identifica a cada usuario en la tabla *Persona* de la base de datos y que debe coincidir con la columna *DNI*. Lo utilizo para resolver consultas en primera persona del tipo "*Cuantas reuniones tengo*...". Editando este archivo se puede ejecutar el sistema para diferentes usuarios sin necesidad de compilar de nuevo el sistema. El formato de este archivo es el mismo que el utilizado para configurar la funcionalidad log y mencionado en el párrafo anterior.

El componente de la base de datos representa el back end del sistema y contiene una clase denominada *UtilitiesDB.java* cuyos métodos utilizo para interactuar con la base de datos. Así hay un método para establecer una conexión con la base de datos, otro método para ejecutar consultas SQL, y finalmente un método para cerrar una conexión con la base de datos. El archivo *db.properties* permite configurar los parámetros de conexión de la base de datos sin necesidad de compilar todo el sistema. Su formato es el mismo que el utilizado para configurar la funcionalidad log.

El último componente representa el front end del sistema y contiene una clase denominada *MainWindow.java* cuyos métodos utilizo para gestionar la interacción del usuario con el sistema. Esta basado en la librería gráfica de Java denominada Swing. Derivando clases predefinidas e implementando métodos predefinidos de Swing consigo implementar el control de las diferentes acciones necesarias para el procesamiento de las consultas introducidas por el usuario, para la ejecución de las consultas SQL y para la presentación y actualización de los resultados de dichas consultas.

Finalmente, el directorio *src* contiene los archivos de especificaciones necesarios para la generación del analizador léxico y el analizador sintáctico. El contenido y la función de estos dos archivos denominados *NQL.jlex* y *NQL.cup* respectivamente, los explico en el apartado que describe el núcleo del sistema NQL.

A continuación tenemos el directorio *db* que contiene dos archivos de comandos SQL para crear el esquema de base de datos en MySQL. El archivo *createDBagenda.sql* lo utilizo para crear la base de datos y el usuario con el cual accederá el sistema a la base de datos que implementa el dominio de aplicación. El archivo *createTables.sql* lo utilizo para crear las diferentes tablas que forman la base de datos. Por otro lado, también contiene un archivo de comandos SQL denominado *populateTables.sql*, para insertar

datos en las tablas de la base de datos y un archivo de comandos SQL denominado *dropTables.sql*, para eliminar todas las tablas de la base de datos. Estos archivos son de gran utilidad durante el desarrollo del sistema ya que permiten crear y modificar los datos necesarias para probar el sistema. Como he mencionado al comienzo de esta sección, el archivo *build.xml* contiene especificaciones para ejecutar estos archivos de comandos SQL mediante la herramienta de compilación automática Ant.

El directorio *doc* contiene archivos con consultas en lenguaje natural y sus correspondientes consultas en SQL. El archivo *questions* contiene todas las consultas en lenguaje natural reconocidas por el sistema. Además, por cada objeto, tabla o relación de la base de datos existe un archivo con consultas relacionadas con los respectivos objetos. En cada uno de ellos se encuentra la asociación entre las consultas en lenguaje natural y las correspondientes consultas en SQL (ver el procedimiento para la implementación de un nuevo dominio de aplicación en el anexo de instalación y ejecución).

El directorio *lib* contiene cuatro librerías. La librería *mysql-connector-java-5.1.18-bin.jar* contiene un conector JDBC para el gestor de base de datos MySQL y es utilizado en tiempo de ejecución. La librería *log4j-1.2.16.jar* contiene las clases utilizadas para añadir la funcionalidad log al sistema NQL. La librería *JLex_1.2.6.jar* contiene el código binario para ejecutar el generador de analizadores léxicos, JLex. La librería *CUP_10k.jar* contiene el código binario para ejecutar el generador de analizadores sintácticos, CUP. A diferencia de las dos anteriores, estas dos últimas librerías no son necesarias ni para la compilación ni para la ejecución del sistema. Las utilizo únicamente para la generación del analizador léxico y para la generación del analizador sintáctico a partir de los archivos *NQL.jlex* y *NQL.cup*, respectivamente. Pudiendo estar fuera de la estructura de desarrollo, se encuentran en la misma por convenencia (ver el archivo *build.xml*).

Por último, en el directorio raíz se encuentra el archivo *nql.log* en el cual se registran todas las acciones y mensajes de error que realiza y emite el sistema NQL.
Como ya he explicado, los archivos *build.properties* y *build.xml* los utilizo para automatizar mediante la herramienta Ant, ciertas tareas de ejecución y compilación. Así, no solo automatizo la compilación de todos los archivos con código fuente Java sino también la ejecución de los generadores *JLex* y *CUP* y la ejecución de los ya mencionados archivos de comandos SQL.
Aunque para ello se hace necesario la instalación de la herramienta Ant, es posible prescindir de un entorno de desarrollo integrado como *Eclipse* o *NetBeans*, facilitando

además la portabilidad de toda la estructura de directorios y archivos entre diferentes entornos.

3.3. Front end. Interfaz gráfico de usuario

Como muestro a continuación, el interfaz de usuario es muy simple y esta formado por una única ventana.

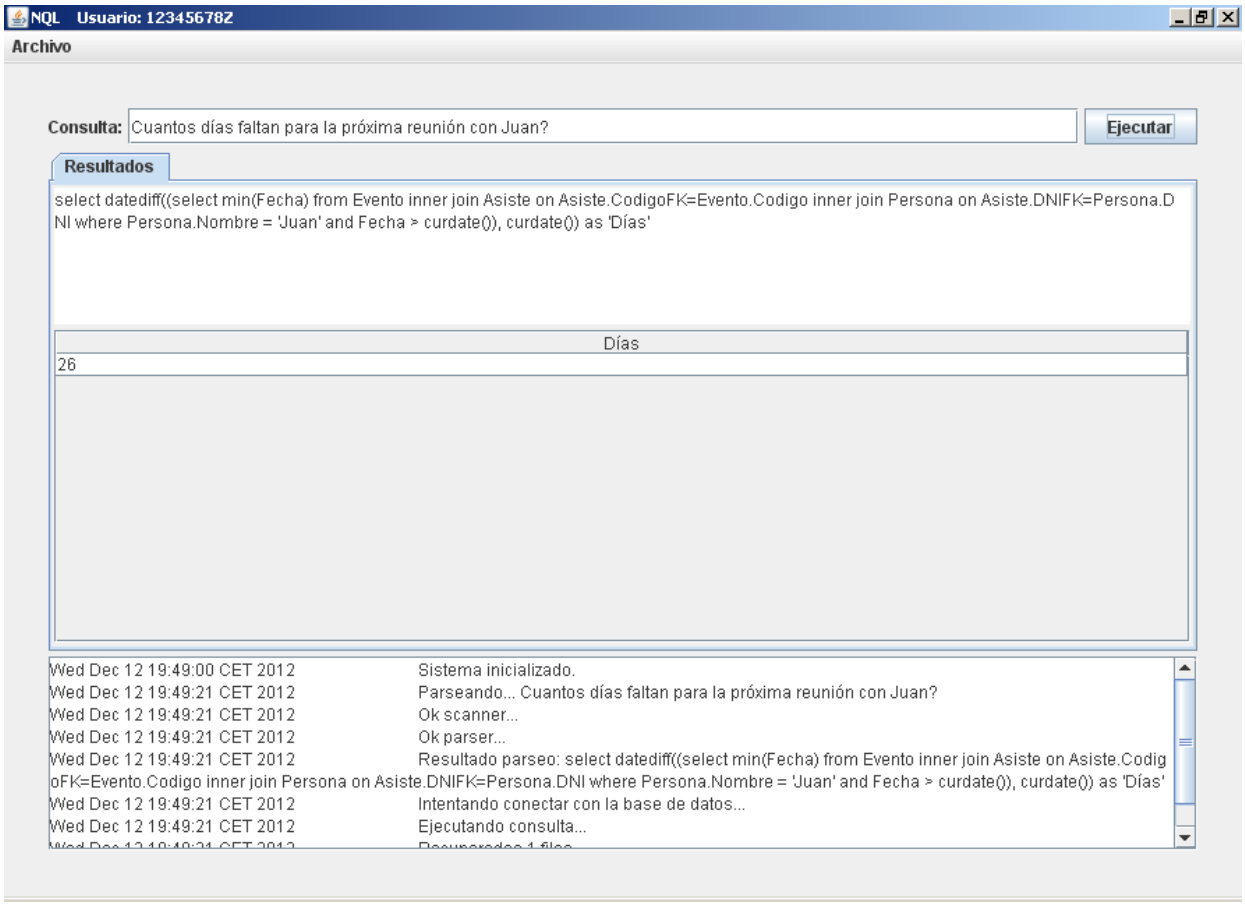


Figura 6. Pantalla del interfaz gráfico de usuario

En la parte del título de la ventana se muestra el identificador del usuario para el cual esta configurado el sistema. Este identificador se corresponde a la clave primaria de la tabla Persona de la base de datos y se configura en el archivo de configuración *nql.properties*. Las consultas se introducen a través de un campo de texto y el procesamiento de la misma se realiza mediante el botón etiquetado como *Ejecutar*. El resultado de la transformación a SQL de la consulta introducida se muestra en un área de texto situado en la parte superior del apartado etiquetado como *Resultados*.

Debajo de la consulta SQL generada se muestra en una tabla el resultado de la ejecución de la misma. En ella se muestran de forma dinámica las columnas resultantes de cada consulta SQL. A continuación del apartado de resultados, en la parte inferior de la ventana, se muestran las diferentes acciones que realiza el sistema durante su inicialización, durante el procesamiento de las consultas introducidas por el usuario y durante la ejecución de cada consulta SQL.

En la parte superior de la ventana existe un menú etiquetado como *Archivo* con la opción para salir del sistema NQL.

El interfaz gráfico de usuario está desarrollado con componentes de la librería gráfica de Java, denominada *Swing*. El código asociado al interfaz está contenido en las clases *NQLMain.java* y *MainWindow.java*. Consiste en un contenedor de tipo JFrame en el cual se incluyen el resto de componentes que forman el interfaz: JMenu para el menú de la parte superior, JTextField para el campo de texto de la consulta en lenguaje natural, JButton para el botón de ejecución, JTable para los mostrar los resultados de las consultas, JScrollPane para mostra los mensajes de acciones, etc...

Lo más destacable del interfaz de usuario es la funcionalidad asociada al botón Ejecutar y la forma que recibe los datos de resultado desde el back end. Además de gestionar mediante eventos de Swing los componente gráficos del interfaz de usuario, la clase *MainWindow.java* contiene el código necesario para controlar la ejecución de cada componente del sistema NQL. Crea instancias para el analizador léxico y el analizador sintáctico y sobre estas, invoca los métodos necesarios para el procesamiento de la consulta introducida por el usuario. Una vez finalizado el procesamiento de la consulta en lenguaje natural y obtenida la correspondiente consulta SQL, crea una instancia del back end invocando el método correspondiente a la ejecución de dicha consulta SQL. Finalmente, muestra los resultados obtenidos de la base de datos a través del back end. De esta forma se puede considerar a la clase *MainWindow.java* como un controlador de los componentes que forman el sistema NQL.

En cuanto a la transferencia de datos entre el back end y front end, el primero transforma los datos resultantes contenidos en una estructura de JDBC denominada ResultSet a un vector de vectores para que el front end pueda mediante una estructura de tipo JTable mostrar los resultados a través del interfaz gráfico.

Por último, quiero mencionar que el código de la clase *MainWindow.java* lo he obtenido

del sistema NLP-Reduce adaptándolo a los requisitos del sistema NQL.

3.4. Núcleo. Analizadores léxico y sintáctico

El procesamiento de las consultas en lenguaje natural se realiza mediante un analizador léxico y un analizador sintáctico generados respectivamente, con JLex y con CUP. Ambos están integrados entre sí formando el núcleo del sistema NQL.

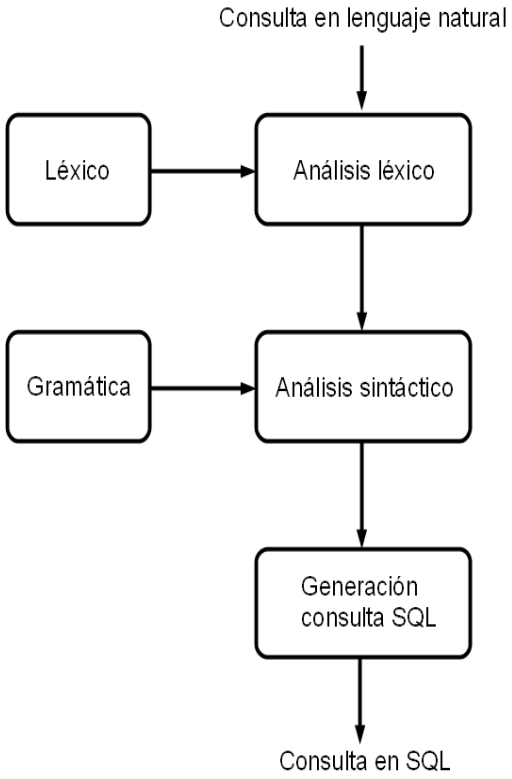


Figura 7. Diagrama de flujo de datos del núcleo del sistema

En la etapa del análisis léxico se procesa la consulta en lenguaje natural en base a un vocabulario o léxico, especificado en el archivo *NQL.jlex*. En esta etapa se determinan las unidades léxicas que forman la consulta. Cada unidad léxica es anotada según una categoría específica formando un token. Cada token es procesado en la siguiente etapa correspondiente al análisis sintáctico. En esta etapa se analiza la estructura del conjunto de tokens que representa la consulta en lenguaje natural. En base a una gramática especificada en el archivo *NQL.cup*, se determina la aceptación de la consulta en lenguaje natural por parte del analizador sintáctico y en su caso, se genera la correspondiente consulta SQL. A continuación describo las especificaciones léxicas y

sintácticas programadas para el sistema NQL.

El archivo *NQL.jlex* (ver anexo 3) contiene las especificaciones léxicas necesarias para que el analizador léxico generado con JLex pueda procesar la consulta en lenguaje natural. Las especificaciones se pueden dividir en tres partes fundamentales.

La **primera parte** contiene código fuente en Java para que el usuario pueda ampliar las prestaciones del analizador léxico generado. Para el sistema NQL utilizo esta parte para definir una simple clase contenedora de la consulta SQL generada por NQL.

La **segunda parte** de las especificaciones léxicas de JLex contienen una serie de directivas y macros para guiar al generador del analizador léxico.
Las directivas que utilizo sirven para integrar JLex con CUP, para denominar y calificar la clase resultante del analizador léxico y para que este reconozca como entrada el juego de caracteres ANSI extendido de 256 caracteres. Esto último es necesario debido a las vocales acentuadas y la letra ñ.
Siguiendo a las directivas declaro tres macros que definen expresiones regulares que el analizador léxico utilizará para reconocer parámetros presentes en las consultas en lenguaje natural, como son, DNI, nombre y apellidos, tema y fechas.

La **tercera parte** de las especificaciones son reglas que representan expresiones regulares que el analizador léxico reconocerá en la entrada y acciones asociadas a cada expresión regular que el analizador léxico ejecutará al reconocerla en la consulta en lenguaje natural.

Como ejemplo, muestro las siguientes reglas que forman parte de las especificaciones léxicas utilizadas para el sistema NQL.

```
"viene"                 { return new Symbol(NQLSym.tk_VIENE); }
"viernes"|"Viernes"     { return new Symbol(NQLSym.tk_VIERNES); }
"última"                { return new Symbol(NQLSym.tk_ULTIMA); }
{dni}                   { return new Symbol(NQLSym.tk_DNIVAL,
                                    new String(yytext())); }
{nom}                   { return new Symbol(NQLSym.tk_NOMVAL,
                                    new String(yytext())); }
{fecha}                 { return new Symbol(NQLSym.tk_FECHA,
                                    new String(yytext())); }
```

Para las primeras tres reglas, el analizador sintáctico crea y devuelve una instancia de la clase *Symbol*, utilizada para la intergación de JLex con CUP e inicializada con una categoría o token para VIENE, VIERNES y ULTIMA, respectivamente. Observar como, utilizando la semántica de las expresiones regulares, se resuelve el reconocimiento de mayúsculas y minúsculas.
Las últimas reglas utilizan macros declaradas anteriormente en las especificaciones para reconocer las expresiones regulares correspondientes al DNI, nombre, apellido y fechas. Como acciones asociadas a estas tres reglas, el analizador léxico crea y devuelve una instancia de la clase Symbol, inicializada en este caso no solo con la categoría de la unidad léxica sino que además incluye el valor de la unidad léxica. Esta es la forma de pasar el valor de los parámetros de entrada al analizador sintáctico. El analizador sintáctico obtiene el valor del parámetro a través de la instancia de *Symbol* pudiendo así procesar dicho valor para generar las consultas SQL correspondientes.
Como se puede ver más adelante en las especificaciones sintácticas para CUP, la misma expresión regular para nombres y apellidos la utilizo para reconocer los temas de los eventos.

El archivo NQL.cup (ver anexo 4) contiene las especificaciones sintácticas necesarias para que el analizador sintáctico pueda procesar la estructura formada por el conjunto de tokens obtenidos del analizador léxico y asociados a cada consulta en lenguaje natural. La parte principal de estas especificaciones la forma una gramática con una estructura similar al ejemplo que describo a continuación.

La notación mostrada corresponde al conjunto de preguntas relacionado con un determinado objeto de la base de datos. Así, *PP* denota el conjunto de preguntas sobre la tabla *Persona* (**P**reguntas**P**ersona). *PE* denota el conjunto de preguntas sobre la tabla *Eventos* (**P**reguntas**E**ventos). *PA* denota el conjunto de preguntas sobre la tabla *Asiste* (**P**reguntas**A**siste).
Mediante el dígito que sigue a estos prefijos identifico cada subconjunto de preguntas en lenguaje natural asociado a una determinada consulta SQL. Recordar qué cada consulta SQL generada tiene asociada un conjunto de consultas en lenguaje natural y que el sistema NQL genera una sola consulta SQL por cada consulta en lenguaje natural.
Así pues, las producciones anotadas de esta forma son las que generan la correspondiente consulta SQL asociada a un subconjunto de consultas en lenguaje natural. Para diferenciar estas producciones de aquellas que reconocen dicho subconjunto, incluyo un dígito *uno* en la tercera parte de la notación.

De esta forma tengo tres tipos de producciones:

- La producción principal
- Las producciones que generan consultas SQL
- Las producciones que reconocen un subconjunto de consultas en lenguaje natural.

```
pregunta ::= PP_1 | PE_1 | PA_1

PP_1 ::= PP_1_1 tk_DOT
PP_1_1 ::= tk_LISTA tk_TODAS tk_LAS tk_PERSONAS
 | tk_MUESTRAME tk_TODAS tk_LAS tk_PERSONAS
 | tk_QUIERO tk_VER tk_TODAS tk_LAS tk_PERSONAS

PE_1 ::= PE_1_1 tk_DOT
PE_1_1 ::= tk_LISTA tk_TODOS tk_LOS tk_EVENTOS
 | tk_MUESTRAME tk_TODOS tk_LOS tk_EVENTOS
 | tk_QUIERO tk_VER tk_TODOS tk_LOS tk_EVENTOS
 | tk_LISTA tk_TODAS tk_LAS tk_REUNIONES
 | tk_MUESTRAME tk_TODAS tk_LAS tk_REUNIONES
 | tk_QUIERO tk_VER tk_TODAS tk_LAS tk_REUNIONES

PA_1 ::= PA_1_1 tk_QST
PA_1_1 ::= tk_TENGO tk_ALGUNA tk_REUNION tk_MANANA tk_POR tk_LA tk_MANANA

tk_DOT ::= '.'
tk_QST ::= '?'
```

La producción principal, *pregunta*, representa el conjunto de consultas en lenguaje natural reconocidas por el sistema NQL y gráficamente se puede representar mediante el siguiente diagrama sintáctico.

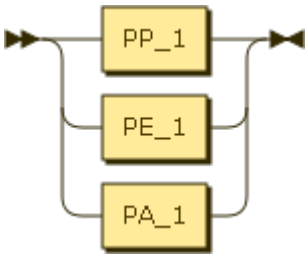


Figura 8. Diagrama sintáctico correspondiente a la producción *pregunta*

El símbolo no terminal *PP_1* en la producción principal hace referencia al conjunto de consultas SQL generado por el sistema NQL sobre la tabla *Persona*. Como ejemplo simplificado, *PP_1* genera la siguiente consulta SQL.

```
select * from Persona
```

Las especificaciones sintácticas de CUP permiten asociar acciones a cada producción. Una acción es un conjunto de sentencias Java. Mediante la composición de una cadena de caracteres se añaden los elementos SQL necesarios para obtener la consulta SELECT apropiada. Aunque a efectos de este ejemplo no lo muestro aquí, en el archivo NQL.cup mostrado en el anexo 4 se pueden observar las acciones que generan cada consulta SQL. Para algunas consultas SQL es necesario obtener parámetros como fechas, nombres y apellidos, temas o DNI de la consulta en lenguaje natural. La integración entre JLex y CUP simplifica mucho el paso de parámetros entre el analizador léxico y el analizador sintáctico. De esta forma, el analizador sintáctico únicamente deberá procesar los valores de los parámetros obtenidos del analizador léxico para adaptarlos a los formatos requeridos por SQL.

A continuación muestro el diagrama sintáctico que representa a la producción *PP_1*.

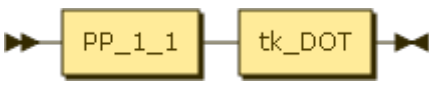


Figura 9. Diagrama sintáctico correspondiente a la producción *PP_1*

El símbolo no terminal *PP_1_1* representa el conjunto de las estructuras sintácticas correspondientes a determinadas consultas en lenguaje natural y formadas cada una de

ellas por un conjunto de símbolos terminales. Estos últimos a su vez representan cada uno de ellos, un token obtenido del analizador léxico.

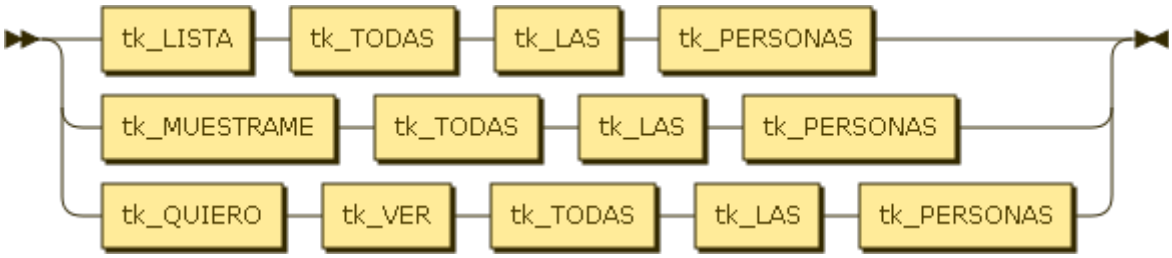


Figura 10. Diagrama sintáctico correspondiente a la producción *PP_1_1*

Como se puede ver en el anterior diagrama, la producción *PP_1_1* solo contiene símbolos terminales, el conjunto de los cuales representa en este caso, tres consultas en lenguaje natural. Observar que estas consultas deben finalizar por un punto debido al símbolo terminal *tk_DOT* presente en la producción *PP_1*.
El resultado de la producción *PP_1* se puede interpretar como la generación de una determinada consulta SQL para un conjunto de tres posibles consultas en lenguaje natural.
Continuando con el ejemplo de gramática simplificada, tenemos la producción *PE_1* que genera la siguiente consulta SQL sobre la tabla *Evento*.

```
select * from Evento
```

A continuación muestro el diagrama sintáctico correspondiente a la producción *PE_1*.

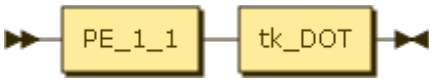


Figura 11. Diagrama sintáctico correspondiente a la producción *PE_1*

El símbolo no terminal *PE_1_1* representa el conjunto de las estructuras sintácticas correspondientes a determinadas consultas en lenguaje natural y formadas cada una de ellas por un conjunto de símbolos terminales como muestro en el siguiente diagrama sintáctico.

Figura 12. Diagrama sintáctico correspondiente a la producción *PE_1_1*

Como se puede ver en el anterior diagrama sintáctico, la consulta SQL generada por la producción *PE_1* esta asociada a seis posible consultas en lenguaje natural.

Completando el ejemplo de gramática simplificada, tenemos finalmente la producción *PA_1* que genera la siguiente consulta SQL sobre la tabla *Asiste*.

```
select (case count(*) when 0 then 'no' else 'si' end) as 'Respuesta'
  from Evento inner join Asiste on Asiste.CodigoFK = Evento.Codigo
  inner join Persona on Asiste.DNIFK=Persona.DNI
where Confirmado = '1' and Fecha = date_add(curdate(), interval 1 day)
  and Hora_inicio < Time('12:00:00') and DNI='{whoami}'
```

A continuación muestro el diagrama sintáctico correspondiente a la producción *PA_1*.

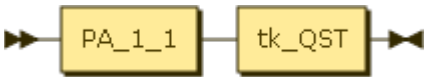


Figura 13. Diagrama sintáctico correspondiente a la producción *PA_1*

El símbolo no terminal *PA_1_1* representa el conjunto de las estructuras sintácticas correspondientes a una determinada consulta en lenguaje natural y formada por un conjunto de símbolos terminales como muestro en el siguiente diagrama sintáctico.

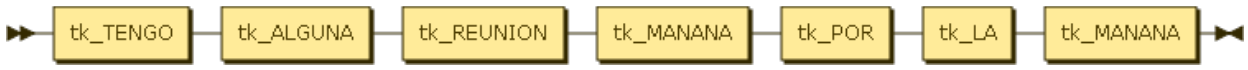


Figura 14. Diagrama sintáctico correspondiente a la producción *PA_1_1*

Como se puede ver en el anterior diagrama sintáctico, la consulta SQL generada por la producción *PA_1* esta asociada a una única consulta en lenguaje natural. Notar también que debido al símbolo terminal *tk_QST* en la producción *PA_1*, la consulta anterior debe finalizar con un carácter de interrogación.

3.4.1. Consultas en primera persona

Sí se observa la última condición de la anterior consulta SQL, se observa un parámetro denominado *whoami*. Se trata de una variable utilizada por el sistema NQL para resolver las consultas en primera persona. Como he explicado en el apartado correspondiente al entorno de desarrollo, el archivo de configuración *nql.properties* almacena el valor del identificador del usuario para el cual fue configurado el sistema. Cada vez que se procesa una consulta en lenguaje natural, el analizador sintáctico obtiene el identificador del usuario almacenando su valor en la variable *whoami*. Cuando el analizador sintáctico debe generar una consulta SQL asociada a una consulta en primera persona, utiliza el valor de esta variable para formar la debida condición en la sentencia SELECT.

3.5. Back end. Base de datos

El principal objetivo del sistema NQL es actuar como interfaz a una base de datos. Así el back end del sistema NQL incluye la implementación de una base de datos que contiene los datos relacionados con el dominio de aplicación que en este caso es una simple gestión de agenda de reuniones. Como he mencionado anteriormente, la implementación de la base de datos la he realizado con el gestor de base de datos MySQL.

En el siguiente diagrama muestro el modelo entidad-relación correspondiente al dominio de aplicación.

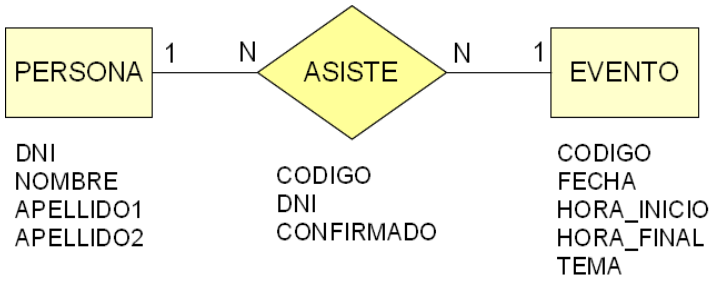


Figura 15. Diagrama del modelo Entidad - Relación

A continuación describo el detalle de cada objeto del modelo que representa el dominio de aplicación. La tabla Persona contiene los datos asociados a cada persona que pueda estar relacionada con alguna reunión o evento y que sean relevantes para el dominio de aplicación. De esta forma se tienen las siguientes columnas para la tabla Persona.

Nombre	Tipo	Clave	Acepta valores *null*
DNI	CHAR(9)	Primaria	no
Nombre	VARCHAR(250)		no
Apellido1	VARCHAR(250)		no
Apellido2	VARCHAR(250)		no

La tabla Evento contiene los datos representativos de cada evento o reunión.

Nombre	Tipo	Clave	Acepta valores *null*
Codigo	MEDIUMINT UNSIGNED	Primaria	no
Fecha	Date		no
Hora_inicio	Time		no
Hora_final	Time		no
Tema	VARCHAR(250)		no

La relación entre las dos entidades anteriores, además de contener los datos necesarios para implementar la misma relación contiene una columna para registrar la confirmación de asistencia de una persona a un evento.

Nombre	Tipo	Clave	Acepta valores *null*
Id	MEDIUMINT UNSIGNED	Primaria	no
DNIFK	CHAR(9)	Extranjera	no
CodigoFK	MEDIUMINT UNSIGNED	Extranjera	no
Confirmado	TINYINT UNSIGNED		no

Como se puede ver en los detalles anteriores, dada la simplicidad del modelo, las columnas de todos los objetos del modelo no aceptan valores vacíos o *null*.

En el apartado correspondiente a la estructura de desarrollo ya he descrito la localización del código necesario para realizar la implementación del anterior modelo sobre un gestor de bases de datos MySQL.

Otra parte fundamental del back end es una capa de código Java que encapsula los detalles de implementación de la base de datos. Esta capa desarrollada sobre el entorno de conectividad de bases de datos en Java (JDBC), la he implementado en la clase *UtilitiesDB.java*. Básicamente, esta clase presenta dos métodos típicos para realizar una conexión y desconexión con una base de datos y un tercer método para ejecutar una consulta SQL sobre una base de datos. Mediante esta capa intento facilitar la portabilidad del sistema NQL entre diferentes gestores de bases de datos.

3.6. Tratamiento de errores

El tratamiento de errores que he programado para NQL es muy básico. Todos los errores contemplados generan un mensaje informativo que es registrado en el log. Algunos errores presentan además un mensaje al usuario a través del interfaz gráfico de usuario.

Contemplo errores de configuración debidos a la ausencia de archivos de configuración o configuraciones erróneas. También contemplo errores de la base de datos. Estos pueden ser debidos a errores relacionados con la conexión o desconexión del back end con el sistema gestor de bases de datos o bien por errores en la ejecución de sentencias SQL. Estos últimos no deberían producirse al haber sidos probados manualmente.

Por último pero no menos importantes, contemplo los errores que se pueden producir durante el procesamiento de la consulta en lenguaje natural. Si el usuario introduce una consulta sintácticamente errónea o una consulta que el sistema no reconoce, el sistema mostrará a través del interfaz gráfico de usuario el siguiente mensaje: "*No entiendo la consulta:* [consulta introducida por el usuario]".

4. Limitaciones y pruebas del sistema

4.1. Limitaciones

Aunque NQL es un sistema NLIDB relativamente limitado, hay algunas limitaciones que se podrían eliminar dedicando más tiempo de investigación y desarrollo del que está determinado por la planificación. Algunas de las limitaciones que se podrían eliminar son:

- El sistema solo admite nombres y apellidos simples de una única palabra.
- El formato de fecha reconocido por el sistema es único: dd/mm/aaaa
- El tema de los eventos reconocidos por el sistema solo puede estar formado por una única palabra.
- Las consultas reconocidas por el sistema siempre deben acabar o bien, por un punto o por un signo de interrogación.
- El sistema no realiza sugerencias o corrección de errores con palabras desconocidas (ver apartado 5.5.1 *Correcciones y sugerencias*).

4.2. Pruebas

Durante el desarrollo del sistema he realizado pruebas unitarias a cada componente. Primero, y siguiendo el orden de desarrollo, he probado el interfaz gráfico de usuario y después el back end. Una vez integrado el front end con el back end he realizado pruebas de su integración.

Después de desarrollar el núcleo e integrarlo con los otros dos componentes principales he realizado pruebas para esta otra integración. Además, he considerado en esta prueba, los componentes secundarios de configuración y el log.

Por otro lado, para cada consulta en lenguaje natural que el sistema NQL reconoce (ver anexo 2) he probado dos factores. Primero, que el sistema reconozca la consulta en lenguaje natural evaluándola de forma léxica y sintáctica. Segundo, para cada una de

las consultas en lenguaje natural reconocidas, que el sistema gestor de base de datos ejecute sin errores las correspondientes consultas SQL generadas. Estas dos pruebas, realizadas en dos etapas han sido fáciles de realizar.

En el código de control asociado al botón *Ejecutar* en la clase *MainWindow.java*, cree un vector de cadenas de caracteres para almacenar el conjunto de consultas en lenguaje natural y un bucle para recorrer dicho vector. Mediante estas dos construcciones proceso, para la primera etapa, todas las consultas en lenguaje natural únicamente con el analizador léxico y con el analizador sintáctico. Desactivando también todas las salidas del sistema exceptuando la del log, ejecuto el sistema y activo el procesamiento mediante el botón *Ejecutar*. En el log pude comprobar la cantidad de consultas en lenguaje natural procesadas con error.

Una vez comprobada la ausencia de errores en la primera etapa y mediante el mismo vector y bucle, para la segunda etapa añadí al procesamiento léxico y sintáctico la ejecución por parte del gestor de base de datos, de cada consulta SQL generada. De esta forma pude comprobar en el log el procesamiento correcto de cada consulta SQL generada por el sistema.

Más difícil de probar fueron los resultados obtenidos de la base de datos al procesarse cada consulta en lenguaje natural. Para verificarlos se debe considerar la presencia y la ausencia de datos relevantes a cada consulta. Dado el dominio de aplicación, las consultas pueden presentar diferentes parámetros, como nombres, fechas, periodos, horas y temas. Así, para probar cada consulta deben existir datos con valores apropiados a cada caso de prueba. En general e independientemente del dominio de aplicación, esta tarea de generación de datos apropiados convierte estas pruebas en un procedimiento muy costoso. Por todo ello y por los limites temporales impuestos para este proyecto, estas pruebas las he realizado de forma no exhaustiva limitándome a algunos casos concretos con valores representativos.

5. Conclusiones

En esta sección expongo las conclusiones alcanzadas con la realización de este proyecto.

5.1. Ampliación

El sistema desarrollado en este proyecto es un sistema muy simple pero puede servir

como referencia a otros sistemas más sofisticados. La arquitectura y los componentes elegidos para su realización permiten considerables ampliaciones y modificaciones. Así por ejemplo, sería posible compactar las especificaciones léxicas y sintácticas para conseguir un léxico y una gramática más genérica. Con ello se ampliarían las posibilidades de portabilidad a otros dominios de aplicación con menores esfuerzos de adaptación y configuración.

5.1.1. Correcciones y sugerencias

De forma resumida, el diseño del sistema NQL se basa en la siguiente idea: Dado un dominio de aplicación y determinadas todas las consultas en lenguaje natural posibles sobre este dominio, relacionar o mapear cada una de ellas con una consulta SQL que obtenga los resultados que den respuesta a la consulta en lenguaje natural.

Así, surgen dos cuestiones fundamentales:
Con 'todas las consultas en lenguaje natural posibles' me refiero a todas aquellas consultas que el sistema NQL deba reconocer. Hay que suponer que se trata de un conjunto finito de consultas. Al ser finito, la cantidad de consultas reconocidas determinará la prestación principal del sistema.

Hay varias formas para implementar la relación entre una consulta en lenguaje natural y una consulta SQL. De forma *naïv*, mediante una tabla. Pero entonces, sería demasiado difícil procesar los parámetros de una consulta, como por ejemplo, nombres, fechas, temas, etc...

Si, además de una tabla, se considera la utilización de expresiones regulares sería posible establecer la relación entre las consultas en lenguaje natural y las consultas SQL obteniendo además los valores asociados a los parámetros de las consultas en lenguaje natural.

Entonces, por qué utilizar un analizador léxico y un analizador sintáctico?

El potencial de utilizar un analizador léxico y un analizador sintáctico radica en su funcionalidad para interceptar palabras desconocidas por el sistema. Desconocidos, o bien, debido a errores léxicos cometidos por el usuario o bien debido a no estar contemplados en las especificaciones del sistema.
De esta forma, cuando el sistema intercepta una palabra desconocida sería posible

comparar esta palabra con todas las palabras que el analizador léxico reconoce.
La comparación se podría realizar en base a una métrica de edición como por ejemplo, la distancia de Levenshtein. Así, no solo se podrían realizar ciertas correcciones sobre la consulta en lenguaje natural sino que además y mediante un parámetro configurable para definir un margen de aproximación entre las palabras desconocidas y las palabras reconocidas por el sistema, se podrían realizar también, sugerencias al usuario por parte del sistema.

5.1.2. Portabilidad

Para portar el sistema NQL a un nuevo entorno se deben tener en cuenta los siguientes aspectos relacionados con el nuevo entorno.

El procedimiento para la **implementación de un nuevo dominio de aplicación** lo describo mediante los siguientes pasos:

1. Especificar el conjunto reconocido de consultas en lenguaje natural.
2. Asociar a cada consulta en lenguaje natural la correspondiente consulta SQL que resuelve la consulta en lenguaje natural.
3. Por cada consulta SQL, agrupar las consultas en lenguaje natural asociadas.
4. Del conjunto de consultas en lenguaje natural, extraer el vocabulario para confeccionar la especificación del analizador léxico JLex (archivo *NQL.jlex*).
5. A partir de la especificación del analizador léxico y la agrupación por consulta SQL de consultas en lenguaje natural, confeccionar la gramática para el analizador sintáctico CUP (archivo *NQL.cup*). Posiblemente y durante este proceso, se deben parametrizar las consultas SQL adecuando en su caso, los valores y formatos generados por el analizador sintáctico a los valores y formatos esperados por SQL.

Para **cambiar de lenguaje natural** se deben adecuar las especificaciones del analizador léxico JLex (archivo *NQL.jlex*) y las especificaciones del analizador sintáctico CUP (archivo *NQL.cup*).

Para **cambiar de gestor de base de datos** se deben adecuar los parámetros de conexión a la base de datos (archivo *db.properties*) y repasar las consultas SQL en la especificación del analizador sintáctico CUP (archivo *NQL.cup*) por si se utilizan elementos no estándar de SQL y propias del gestor de bases de datos en cuestión.

Para **cambiar de usuario** se debe configurar el identificador que identifica de forma única al usuario en la base de datos en el archivo *nql.properties*.

5.2. Valoración personal

En cuanto a la realización del proyecto en sí, ha resultado interesante el siguiente aspecto. Sin conocimientos sobre el área del procesamiento del lenguaje natural y dada su envergadura, me ha resultado muy difícil encontrar los limites apropiados del alcance del proyecto. Investigando recursos en la red es muy fácil perder demasiado tiempo si no se tiene claro que, dado el nivel académico de este proyecto, no se trata de un proyecto de investigación. Aunque la confusión inicial debida a la cantidad de material disponible me ha conducido a un retraso en el diseño de la arquitectura del sistema NQL, finalmente otro aspecto interesante me ha permitido concluir el desarrollo en los plazos previstos. Y es que quizás el tiempo invertido inicialmente en el paseo por técnicas más o menos avanzadas de procesamiento de lenguaje natural, me ha llevado a buscar la simplicidad de un sistema acorde con los conocimientos que dispongo dado mi nivel académico. Resumiendo, he necesitado encontrar un equilibrio entre investigación y aplicación de conocimientos ya adquiridos durante los estudios para hallar el camino que me ha llevado a la solución presentada en este proyecto.

5.3. Conclusión final

A través del desarrollo del sistema NQL presentado en este proyecto he conseguido reforzar los conocimientos sobre las herramientas de desarrollo de compiladores JLex y CUP. Además de aprender más sobre SQL y específicamente, funciones de MySQL, he adquirido también conocimientos acerca del procesamiento del lenguaje natural. Por un lado he explorado el panorama de los sistemas NLIDB conociendo sus principales ventajas e inconvenientes y por otro lado, he investigado los problemas de la ambigüedad en el procesamiento del lenguaje natural. Con respecto a esto último he sabido que cuanta más información procese un sistema, mejor afrontará las dificultades que presenta la ambigüedad del lenguaje natural.

Con el sistema desarrollado en este proyecto, he comprendido que limitando el dominio de aplicación es posible conseguir cierta eficacia y utilidad de un sistema NLIDB.

6. Referencias

Natural Language Interfaces to Databases – An Introduction
I. Androutsopoulos, G.D. Ritchie
Department of Artificial Intelligence, University of Edinburgh
P. Thanisch
Department of Computer Science, University of Edinburgh
http://arxiv.org/pdf/cmp-lg/9503016.pdf

Natural Language Interfaces to Databases
Ann Copestake, Karen Sparck Jones
Computer Laboratory, University of Cambridge
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.7136&rep=rep1&type=pdf

NLBean version 5: A Natural Language Interface for Databases
Mark Watson, 1997
http://www.markwatson.com/opensource/

NLP-Reduce: A "naive" but Domain-independent Natural Language Interface for Querying Ontologies
Esther Kaufmann, Abraham Bernstein, and Lorenz Fischer, 2007
Universidad de Zürich
https://files.ifi.uzh.ch/ddis/oldweb/ddis/research/talking-to-the-semantic-web/nlpreduce/index.html

SQ-HAL: Natural Language to SQL Translator
Supun Ruwanpura, 2000
Universidad de Monash
http://www.csse.monash.edu.au/hons/projects/2000/Supun.Ruwanpura/

Ambiguity Management in Natural Language Generation
Francis Chantree
Department of Maths and Computing
The Open University
http://www.geocities.ws/fchantree/Chantree_CLUK04.pdf

SQL
http://en.wikipedia.org/wiki/SQL

Select (SQL)
http://en.wikipedia.org/wiki/Select_%28SQL%29

JLex: A Lexical Analyzer Generator for Java
http://www.cs.princeton.edu/~appel/modern/java/JLex/

CUP Parser Generator for Java
http://www.cs.princeton.edu/~appel/modern/java/CUP/

Finding "Next Monday" using MySQL Dates
http://www.gizmola.com/blog/archives/99-Finding-Next-Monday-using-MySQL-Dates.html

7. Anexos

Anexo I. Instalación y ejecución

Para instalar el sistema NQL es suficiente con descomprimir el archivo de distribución en un directorio determinado por el usuario o administrador del sistema.

En el caso de no existir la base de datos que soporta el dominio de aplicación se debe crear primero la base de datos y un usuario para la misma. Una vez creada la base de datos y el usuario, hay que crear las tablas e insertar los datos correspondientes al dominio de aplicación.

Cuando la base de datos ya está disponible se debe configurar el archivo *db.properties* ajustando los parámetros para que el sistema pueda acceder a la base de datos.
Finalmente hay que configurar en el archivo *nql.properties* el identificador único que identifica en la base de datos al usuario que va a interactuar con el sistema NQL.

Una vez instalado y configurado el sistema NQL es posible ejecutarlo desde la línea de comandos o desde un terminal situado en la raíz del directorio de distribución mediante la siguiente orden:

```
java -Djava.ext.dirs=lib -cp bin nql.NQLMain
```

Anexo II. Consultas reconocidas

1. Tengo alguna reunión mañana por la mañana?
2. Tengo alguna reunión mañana?
3. Qué reuniones tengo mañana por la mañana?
4. Qué reuniones tengo mañana?
5. Cuantas personas han confirmado su asistencia a la reunión de hoy?
6. Cuantas personas han confirmado su asistencia a la reunión de mañana?
7. Cuantas personas confirmaron su asistencia a la reunión de ayer?
8. Cuantas personas han confirmado su asistencia a la reunión del lunes de la semana que viene?
9. Cuantas personas han confirmado su asistencia a la reunión del martes de la semana que viene?
10. Cuantas personas han confirmado su asistencia a la reunión del miércoles de la semana que viene?
11. Cuantas personas han confirmado su asistencia a la reunión del jueves de la semana que viene?
12. Cuantas personas han confirmado su asistencia a la reunión del viernes de la semana que viene?
13. Cuantos días faltan para la próxima reunión con Juan?
14. Cuantas reuniones sobre gastos haremos este mes?
15. Cuantas reuniones sobre impuestos haremos esta semana?
16. Lista todos los eventos.
17. Muéstrame todos los eventos.
18. Quiero ver todos los eventos.
19. Lista todas las reuniones.
20. Muéstrame todas las reuniones.
21. Quiero ver todas las reuniones.
22. A que hora comienza la próxima reunión sobre impuestos?
23. A que hora empieza la próxima reunión sobre impuestos?
24. A que hora comenzará la próxima reunión sobre impuestos?
25. A que hora empezará la próxima reunión sobre impuestos?
26. A que hora comienza la reunión sobre impuestos?
27. A que hora empieza la reunión sobre impuestos?
28. A que hora comenzará la reunión sobre impuestos?
29. A que hora empezará la reunión sobre impuestos?
30. A que hora acaba la próxima reunión sobre impuestos?

31. A que hora acabará la próxima reunión sobre impuestos?
32. A que hora finalizará la próxima reunión sobre impuestos?
33. A que hora acaba la reunión sobre impuestos?
34. A que hora acabará la reunión sobre impuestos?
35. A que hora finalizará la reunión sobre impuestos?
36. A que hora comienza la próxima reunión del día 12/12/2012?
37. A que hora empieza la próxima reunión del día 12/12/2012?
38. A que hora comenzará la próxima reunión del día 12/12/2012?
39. A que hora empezará la próxima reunión del día 12/12/2012?
40. A que hora comienza la reunión del día 12/12/2012?
41. A que hora empieza la reunión del día 12/12/2012?
42. A que hora comenzará la reunión del día 12/12/2012?
43. A que hora empezará la reunión del día 12/12/2012?
44. A que hora acaba la próxima reunión del día 12/12/2012?
45. A que hora acabará la próxima reunión del día 12/12/2012?
46. A que hora finalizará la próxima reunión del día 12/12/2012?
47. A que hora acaba la reunión del día 12/12/2012?
48. A que hora acabará la reunión del día 12/12/2012?
49. A que hora finalizará la reunión del día 12/12/2012?
50. A que hora terminó la pasada reunión sobre impuestos?
51. A que hora finalizó la pasada reunión sobre impuestos?
52. A que hora acabó la pasada reunión sobre impuestos?
53. A que hora terminó la última reunión sobre impuestos?
54. A que hora finalizó la última reunión sobre impuestos?
55. A que hora acabó la última reunión sobre impuestos?
56. A que hora terminó la reunión sobre impuestos?
57. A que hora finalizó la reunión sobre impuestos?
58. A que hora acabó la reunión sobre impuestos?
59. Cuando fue la última reunión?
60. Cuando fue la última reunión sobre impuestos?
61. Cuando es la próxima reunión?
62. Cuando es la próxima reunión sobre impuestos?
63. Cuantas reuniones hay la semana que viene?
64. Que reuniones hay la semana que viene?
65. Cuales son las reuniones del lunes de la semana que viene?
66. Cuales son las reuniones del martes de la semana que viene?
67. Cuales son las reuniones del miércoles de la semana que viene?

68. Cuales son las reuniones del jueves de la semana que viene?
69. Cuales son las reuniones del viernes de la semana que viene?
70. Cuantos días faltan para la próxima reunión sobre impuestos?
71. Cuantos días faltan para la próxima reunión?
72. Hay alguna reunión la semana que viene?
73. Lista todas las personas.
74. Muéstrame todas las personas.
75. Quiero ver todas las personas.
76. Como es el primer apellido de la persona con DNI 12345678Z?
77. Como es el primer apellido de la persona con DNI igual a 12345678Z?
78. Como es el primer apellido de la persona cuyo DNI es 12345678Z?
79. Como es el primer apellido de la persona cuyo DNI es igual a 12345678Z?
80. Cual es el primer apellido de la persona con DNI 12345678Z?
81. Cual es el primer apellido de la persona con DNI igual a 12345678Z?
82. Cual es el primer apellido de la persona cuyo DNI es 12345678Z?
83. Cual es el primer apellido de la persona cuyo DNI es igual a 12345678Z?
84. Como es el segundo apellido de la persona con DNI 12345678Z?
85. Como es el segundo apellido de la persona con DNI igual a 12345678Z?
86. Como es el segundo apellido de la persona cuyo DNI es 12345678Z?
87. Como es el segundo apellido de la persona cuyo DNI es igual a 12345678Z?
88. Cual es el segundo apellido de la persona con DNI 12345678Z?
89. Cual es el segundo apellido de la persona con DNI igual a 12345678Z?
90. Cual es el segundo apellido de la persona cuyo DNI es 12345678Z?
91. Cual es el segundo apellido de la persona cuyo DNI es igual a 12345678Z?
92. Como son los apellidos de la persona con DNI 12345678Z?
93. Como son los apellidos de la persona con DNI igual a 12345678Z?
94. Como son los apellidos de la persona cuyo DNI es 12345678Z?
95. Como son los apellidos de la persona cuyo DNI es igual a 12345678Z?
96. Cuales son los apellidos de la persona con DNI 12345678Z?
97. Cuales son los apellidos de la persona con DNI igual a 12345678Z?
98. Cuales son los apellidos de la persona cuyo DNI es 12345678Z?
99. Cuales son los apellidos de la persona cuyo DNI es igual a 12345678Z?
100. Hay alguien que se llame Juan Pastor?
101. Hay alguna persona que se llame Juan Pastor?
102. Hay alguien con nombre Juan?
103. Hay alguien cuyo nombre es Juan?
104. Hay alguien que de nombre se llame Juan?

105. Hay alguien que se llame Juan de nombre?
106. Hay alguien que se llame Juan?
107. Hay alguna persona con nombre Juan?
108. Hay alguna persona cuyo nombre es Juan?
109. Hay alguna persona que se llame Juan?
110. Hay alguna persona que se llame Juan de nombre?

Anexo III. Léxico JLex: archivo NQL.jlex

```
package nql;

import java_cup.runtime.Symbol;

class SQLStringBuffer {
	private StringBuffer result;

	public SQLStringBuffer() {
		super();
		this.result = new StringBuffer();
	}

	public StringBuffer getResult() {
		return result;
	}

	public void clearResult() {
		this.result = new StringBuffer();
	}

	public void setResult(StringBuffer result) {
		this.result = result;
	}

	@Override
	public String toString() {
		StringBuilder builder = new StringBuilder();
		builder.append(result);
		return builder.toString();
	}
}

%%
%class NQLScanner
%public
%cup
%full
dni=[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][A-Z]
nom=[A-Za-záéíóúñ]+
fecha=(0[1-9]|[12][0-9]|3[01])[/](0[1-9]|1[012])[/](19|20|21)[0-9][0-9]
%%
\.                      { return new Symbol(NQLSym.tk_DOT); }
\?                      { return new Symbol(NQLSym.tk_QST); }
"Cual"                  { return new Symbol(NQLSym.tk_CUAL); }
"Cuales"                { return new Symbol(NQLSym.tk_CUALES); }
"Cuando"                { return new Symbol(NQLSym.tk_CUANDO); }
"Cuantas"               { return new Symbol(NQLSym.tk_CUANTAS); }
"Cuantos"               { return new Symbol(NQLSym.tk_CUANTOS); }
"DNI"                   { return new Symbol(NQLSym.tk_DNI); }
"Lista"                 { return new Symbol(NQLSym.tk_LISTA); }
"Muéstrame"             { return new Symbol(NQLSym.tk_MUESTRAME); }
"Quiero"                { return new Symbol(NQLSym.tk_QUIERO); }
"a"|"A"                 { return new Symbol(NQLSym.tk_A); }
"acaba"                 { return new Symbol(NQLSym.tk_ACABA); }
"acabará"               { return new Symbol(NQLSym.tk_ACABARA); }
"acabó"                 { return new Symbol(NQLSym.tk_ACABO); }
```

```
"alguien"               { return new Symbol(NQLSym.tk_ALGUIEN); }
"alguna"                { return new Symbol(NQLSym.tk_ALGUNA); }
"apellido"              { return new Symbol(NQLSym.tk_APELLIDO); }
"apellidos"             { return new Symbol(NQLSym.tk_APELLIDOS); }
"asistencia"            { return new Symbol(NQLSym.tk_ASISTENCIA); }
"ayer"                  { return new Symbol(NQLSym.tk_AYER); }
"comenzará"             { return new Symbol(NQLSym.tk_COMENZARA); }
"comienza"              { return new Symbol(NQLSym.tk_COMIENZA); }
"como"|"Como"           { return new Symbol(NQLSym.tk_COMO); }
"con"                   { return new Symbol(NQLSym.tk_CON); }
"confirmado"            { return new Symbol(NQLSym.tk_CONFIRMADO); }
"confirmaron"           { return new Symbol(NQLSym.tk_CONFIRMARON); }
"cuyo"                  { return new Symbol(NQLSym.tk_CUYO); }
"de"                    { return new Symbol(NQLSym.tk_DE); }
"del"                   { return new Symbol(NQLSym.tk_DEL); }
"día"                   { return new Symbol(NQLSym.tk_DIA); }
"días"                  { return new Symbol(NQLSym.tk_DIAS); }
"el"                    { return new Symbol(NQLSym.tk_EL); }
"empezará"              { return new Symbol(NQLSym.tk_EMPEZARA); }
"empieza"               { return new Symbol(NQLSym.tk_EMPIEZA); }
"es"                    { return new Symbol(NQLSym.tk_ES); }
"esta"                  { return new Symbol(NQLSym.tk_ESTA); }
"este"                  { return new Symbol(NQLSym.tk_ESTE); }
"eventos"               { return new Symbol(NQLSym.tk_EVENTOS); }
"faltan"                { return new Symbol(NQLSym.tk_FALTAN); }
"finalizará"            { return new Symbol(NQLSym.tk_FINALIZARA); }
"finalizó"              { return new Symbol(NQLSym.tk_FINALIZO); }
"fue"                   { return new Symbol(NQLSym.tk_FUE); }
"han"                   { return new Symbol(NQLSym.tk_HAN); }
"haremos"               { return new Symbol(NQLSym.tk_HAREMOS); }
"hay"|"Hay"             { return new Symbol(NQLSym.tk_HAY); }
"hora"                  { return new Symbol(NQLSym.tk_HORA); }
"hoy"                   { return new Symbol(NQLSym.tk_HOY); }
"igual"                 { return new Symbol(NQLSym.tk_IGUAL); }
"jueves"|"Jueves"       { return new Symbol(NQLSym.tk_JUEVES); }
"la"                    { return new Symbol(NQLSym.tk_LA); }
"las"                   { return new Symbol(NQLSym.tk_LAS); }
"llame"                 { return new Symbol(NQLSym.tk_LLAME); }
"los"                   { return new Symbol(NQLSym.tk_LOS); }
"lunes"|"Lunes"         { return new Symbol(NQLSym.tk_LUNES); }
"martes"|"Martes"       { return new Symbol(NQLSym.tk_MARTES); }
"mañana"                { return new Symbol(NQLSym.tk_MANANA); }
"mes"                   { return new Symbol(NQLSym.tk_MES); }
"miércoles"|"Miércoles" { return new Symbol(NQLSym.tk_MIERCOLES); }
"nombre"                { return new Symbol(NQLSym.tk_NOMBRE); }
"para"                  { return new Symbol(NQLSym.tk_PARA); }
"pasada"                { return new Symbol(NQLSym.tk_PASADA); }
"persona"               { return new Symbol(NQLSym.tk_PERSONA); }
"personas"              { return new Symbol(NQLSym.tk_PERSONAS); }
"por"                   { return new Symbol(NQLSym.tk_POR); }
"primer"                { return new Symbol(NQLSym.tk_PRIMER); }
"próxima"               { return new Symbol(NQLSym.tk_PROXIMA); }
"que"|"Que"|"qué"|"Qué" { return new Symbol(NQLSym.tk_QUE); }
"reuniones"             { return new Symbol(NQLSym.tk_REUNIONES); }
"reunión"               { return new Symbol(NQLSym.tk_REUNION); }
"se"                    { return new Symbol(NQLSym.tk_SE); }
"segundo"               { return new Symbol(NQLSym.tk_SEGUNDO); }
"semana"                { return new Symbol(NQLSym.tk_SEMANA); }
"sobre"                 { return new Symbol(NQLSym.tk_SOBRE); }
```

```
"son"                   { return new Symbol(NQLSym.tk_SON); }
"su"                    { return new Symbol(NQLSym.tk_SU); }
"tengo"|"Tengo"         { return new Symbol(NQLSym.tk_TENGO); }
"terminó"               { return new Symbol(NQLSym.tk_TERMINO); }
"todas"                 { return new Symbol(NQLSym.tk_TODAS); }
"todos"                 { return new Symbol(NQLSym.tk_TODOS); }
"ver"                   { return new Symbol(NQLSym.tk_VER); }
"viene"                 { return new Symbol(NQLSym.tk_VIENE); }
"viernes"|"Viernes"     { return new Symbol(NQLSym.tk_VIERNES); }
"última"                { return new Symbol(NQLSym.tk_ULTIMA); }
{dni}                   { return new Symbol(NQLSym.tk_DNIVAL,
                                            new String(yytext())); }
{nom}                   { return new Symbol(NQLSym.tk_NOMVAL,
                                            new String(yytext())); }
{fecha}                 { return new Symbol(NQLSym.tk_FECHA,
                                            new String(yytext())); }
[ \t\r\n\f]             { /* ignore white space. */ }
.                       { return new Symbol(NQLSym.tk_ERRLEX,
                                            new String(yytext())); }
```

Anexo IV. Gramática CUP: archivo NQL.cup

```
package nql;


import java_cup.runtime.Symbol;
import org.apache.log4j.Logger;
import java.util.Calendar;
import java.util.GregorianCalendar;

parser code {:

 static Logger log = Logger.getLogger(NQLMain.class.getName());

 static SQLStringBuffer resultSQL = new SQLStringBuffer();
 static String whoami;

 public void setWhoami(String user) {
      whoami = user;
 }

 public String getSQLString() {
      return resultSQL.toString();
 }

 public void report_fatal_error(String message, Object info) throws
java.lang.Exception {
    /* stop parsing (not really necessary since we throw an exception, but) */
    done_parsing();

    /* use the normal error message reporting to put out the message */
    //report_error(message, info);

    /* throw an exception */
    throw new Exception("Can't recover from previous error(s)");
 }

 public void syntax_error(Symbol cur_token) {
    Calendar cal = new GregorianCalendar(); // get current time
    log.error(cal.getTime().toString() + "\tSyntax error");
    //report_error("Syntax error", cur_token);
 }

 public void unrecovered_syntax_error(Symbol cur_token) throws
java.lang.Exception {
    Calendar cal = new GregorianCalendar(); // get current time
    log.error(cal.getTime().toString() + "\tCouldn't repair and continue
parse");
    report_fatal_error("Couldn't repair and continue parse", cur_token);
 }

:}

terminal tk_DOT, tk_QST;
terminal tk_A, tk_ACABA, tk_ACABARA, tk_ACABO, tk_ALGUIEN;
terminal tk_ALGUNA, tk_APELLIDO, tk_APELLIDOS, tk_ASISTENCIA, tk_AYER;
terminal tk_COMENZARA, tk_COMIENZA, tk_COMO, tk_CON, tk_CONFIRMADO;
terminal tk_CONFIRMARON, tk_CUAL, tk_CUALES, tk_CUANDO, tk_CUANTAS;
terminal tk_CUANTOS, tk_CUYO, tk_DE, tk_DEL, tk_DIA;
```

```
terminal tk_DIAS, tk_DNI, tk_EL, tk_EMPEZARA, tk_EMPIEZA;
terminal tk_ES, tk_ESTA, tk_ESTE, tk_EVENTOS, tk_FALTAN;
terminal tk_FINALIZARA, tk_FINALIZO, tk_FUE, tk_HAN, tk_HAREMOS;
terminal tk_HAY, tk_HORA, tk_HOY, tk_IGUAL, tk_JUEVES;
terminal tk_LA, tk_LAS, tk_LISTA, tk_LLAME, tk_LOS;
terminal tk_LUNES, tk_MANANA, tk_MARTES, tk_MES, tk_MIERCOLES;
terminal tk_MUESTRAME, tk_NOMBRE, tk_PARA, tk_PASADA, tk_PERSONA;
terminal tk_PERSONAS, tk_POR, tk_PRIMER, tk_PROXIMA, tk_QUE;
terminal tk_QUIERO, tk_REUNION, tk_REUNIONES, tk_SE, tk_SEGUNDO;
terminal tk_SEMANA, tk_SOBRE, tk_SON, tk_SU, tk_TENGO;
terminal tk_TERMINO, tk_TODAS, tk_TODOS, tk_ULTIMA, tk_VER;
terminal tk_VIENE, tk_VIERNES;
terminal tk_ERRLEX;

terminal String tk_DNIVAL;
terminal String tk_NOMVAL;
terminal String tk_FECHA;

non terminal pregunta;

non terminal PP_1, PP_1_1;
non terminal PP_2;
non terminal String PP_2_1;
non terminal PP_3;
non terminal String PP_3_1;
non terminal PP_4;
non terminal String PP_4_1;
non terminal PP_25;
non terminal String PP_25_1;
non terminal PP_26;
non terminal String PP_26_1;

non terminal PE_1, PE_1_1;
non terminal PE_2;
non terminal String PE_2_1;
non terminal PE_3;
non terminal String PE_3_1;
non terminal PE_4;
non terminal String PE_4_1;
non terminal PE_5;
non terminal String PE_5_1;
non terminal PE_6;
non terminal String PE_6_1;
non terminal PE_7, PE_7_1;
non terminal PE_8;
non terminal String PE_8_1;
non terminal PE_9, PE_9_1;
non terminal PE_10;
non terminal String PE_10_1;
non terminal PE_11, PE_11_1;
non terminal PE_12, PE_12_1;
non terminal PE_13, PE_13_1;
non terminal PE_14, PE_14_1;
non terminal PE_15, PE_15_1;
non terminal PE_16, PE_16_1;
non terminal PE_17, PE_17_1;
non terminal PE_18;
non terminal String PE_18_1;
non terminal PE_19, PE_19_1;
```

```
non terminal PE_20, PE_20_1;

non terminal PA_1, PA_1_1;
non terminal PA_2, PA_2_1;
non terminal PA_3, PA_3_1;
non terminal PA_4, PA_4_1;
non terminal PA_5, PA_5_1;
non terminal PA_6, PA_6_1;
non terminal PA_7, PA_7_1;
non terminal PA_8, PA_8_1;
non terminal PA_9, PA_9_1;
non terminal PA_10, PA_10_1;
non terminal PA_11, PA_11_1;
non terminal PA_12, PA_12_1;
non terminal PA_13;
non terminal String PA_13_1;
non terminal PA_14;
non terminal String PA_14_1;
non terminal PA_15;
non terminal String PA_15_1;

pregunta ::= PP_1 | PP_2 | PP_3 | PP_4 | PP_25 | PP_26
       | PE_1 | PE_2 | PE_3 | PE_4 | PE_5 | PE_6 | PE_7 | PE_8 | PE_9 | PE_10
       | PE_11 | PE_12 | PE_13 | PE_14 | PE_15 | PE_16 | PE_17 | PE_18 | PE_19 |
PE_20
       | PA_1 | PA_2 | PA_3 | PA_4 | PA_5 | PA_6 | PA_7 | PA_8 | PA_9 | PA_10
       | PA_11 | PA_12 | PA_13 | PA_14 | PA_15
;

PP_1 ::= PP_1_1 tk_DOT
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select * from Persona");
:}
;
PP_1_1 ::= tk_LISTA tk_TODAS tk_LAS tk_PERSONAS
            | tk_MUESTRAME tk_TODAS tk_LAS tk_PERSONAS
            | tk_QUIERO tk_VER tk_TODAS tk_LAS tk_PERSONAS
;

PP_2 ::= PP_2_1:d tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select Apellido1 as "+
      "\'Primer apellido\' from Persona where DNI=\'" + d + "\'");
:}
;
PP_2_1 ::= tk_COMO tk_ES tk_EL tk_PRIMER tk_APELLIDO tk_DE tk_LA tk_PERSONA
tk_CON tk_DNI tk_DNIVAL:d1
{:
      RESULT = new String(d1);
:}
            | tk_COMO tk_ES tk_EL tk_PRIMER tk_APELLIDO tk_DE tk_LA tk_PERSONA
tk_CON tk_DNI tk_IGUAL tk_A tk_DNIVAL:d2
{:
      RESULT = new String(d2);
:}
            | tk_COMO tk_ES tk_EL tk_PRIMER tk_APELLIDO tk_DE tk_LA tk_PERSONA
tk_CUYO tk_DNI tk_ES tk_DNIVAL:d3
```

```
{:
      RESULT = new String(d3);
:}
            | tk_COMO tk_ES tk_EL tk_PRIMER tk_APELLIDO tk_DE tk_LA tk_PERSONA
tk_CUYO tk_DNI tk_ES tk_IGUAL tk_A tk_DNIVAL:d4
{:
      RESULT = new String(d4);
:}
            | tk_CUAL tk_ES tk_EL tk_PRIMER tk_APELLIDO tk_DE tk_LA tk_PERSONA
tk_CON tk_DNI tk_DNIVAL:d5
{:
      RESULT = new String(d5);
:}
            | tk_CUAL tk_ES tk_EL tk_PRIMER tk_APELLIDO tk_DE tk_LA tk_PERSONA
tk_CON tk_DNI tk_IGUAL tk_A tk_DNIVAL:d6
{:
      RESULT = new String(d6);
:}
            | tk_CUAL tk_ES tk_EL tk_PRIMER tk_APELLIDO tk_DE tk_LA tk_PERSONA
tk_CUYO tk_DNI tk_ES tk_DNIVAL:d7
{:
      RESULT = new String(d7);
:}
            | tk_CUAL tk_ES tk_EL tk_PRIMER tk_APELLIDO tk_DE tk_LA tk_PERSONA
tk_CUYO tk_DNI tk_ES tk_IGUAL tk_A tk_DNIVAL:d8
{:
      RESULT = new String(d8);
:}
;

PP_3 ::= PP_3_1:d tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select Apellido2 as "+
      "\'Segundo apellido\' from Persona where DNI=\'" + d + "\'");
:}
;
PP_3_1 ::= tk_COMO tk_ES tk_EL tk_SEGUNDO tk_APELLIDO tk_DE tk_LA tk_PERSONA
tk_CON tk_DNI tk_DNIVAL:d1
{:
      RESULT = new String(d1);
:}
            | tk_COMO tk_ES tk_EL tk_SEGUNDO tk_APELLIDO tk_DE tk_LA tk_PERSONA
tk_CON tk_DNI tk_IGUAL tk_A tk_DNIVAL:d2
{:
      RESULT = new String(d2);
:}
            | tk_COMO tk_ES tk_EL tk_SEGUNDO tk_APELLIDO tk_DE tk_LA tk_PERSONA
tk_CUYO tk_DNI tk_ES tk_DNIVAL:d3
{:
      RESULT = new String(d3);
:}
            | tk_COMO tk_ES tk_EL tk_SEGUNDO tk_APELLIDO tk_DE tk_LA tk_PERSONA
tk_CUYO tk_DNI tk_ES tk_IGUAL tk_A tk_DNIVAL:d4
{:
      RESULT = new String(d4);
:}
            | tk_CUAL tk_ES tk_EL tk_SEGUNDO tk_APELLIDO tk_DE tk_LA tk_PERSONA
tk_CON tk_DNI tk_DNIVAL:d5
```

```
{:
      RESULT = new String(d5);
:}
            | tk_CUAL tk_ES tk_EL tk_SEGUNDO tk_APELLIDO tk_DE tk_LA tk_PERSONA
tk_CON tk_DNI tk_IGUAL tk_A tk_DNIVAL:d6
{:
      RESULT = new String(d6);
:}
            | tk_CUAL tk_ES tk_EL tk_SEGUNDO tk_APELLIDO tk_DE tk_LA tk_PERSONA
tk_CUYO tk_DNI tk_ES tk_DNIVAL:d7
{:
      RESULT = new String(d7);
:}
            | tk_CUAL tk_ES tk_EL tk_SEGUNDO tk_APELLIDO tk_DE tk_LA tk_PERSONA
tk_CUYO tk_DNI tk_ES tk_IGUAL tk_A tk_DNIVAL:d8
{:
      RESULT = new String(d8);
:}
;

PP_4 ::= PP_4_1:d tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select Apellido1 as \'Primer
apellido\', "+
      "Apellido2 as \'Segundo apellido\' from Persona where DNI=\'" + d + "\'");
:}
;
PP_4_1 ::= tk_COMO tk_SON tk_LOS tk_APELLIDOS tk_DE tk_LA tk_PERSONA tk_CON
tk_DNI tk_DNIVAL:d1
{:
      RESULT = new String(d1);
:}
            | tk_COMO tk_SON tk_LOS tk_APELLIDOS tk_DE tk_LA tk_PERSONA tk_CON
tk_DNI tk_IGUAL tk_A tk_DNIVAL:d2
{:
      RESULT = new String(d2);
:}
            | tk_COMO tk_SON tk_LOS tk_APELLIDOS tk_DE tk_LA tk_PERSONA tk_CUYO
tk_DNI tk_ES tk_DNIVAL:d3
{:
      RESULT = new String(d3);
:}
            | tk_COMO tk_SON tk_LOS tk_APELLIDOS tk_DE tk_LA tk_PERSONA tk_CUYO
tk_DNI tk_ES tk_IGUAL tk_A tk_DNIVAL:d4
{:
      RESULT = new String(d4);
:}
            | tk_CUALES tk_SON tk_LOS tk_APELLIDOS tk_DE tk_LA tk_PERSONA tk_CON
tk_DNI tk_DNIVAL:d5
{:
      RESULT = new String(d5);
:}
            | tk_CUALES tk_SON tk_LOS tk_APELLIDOS tk_DE tk_LA tk_PERSONA tk_CON
tk_DNI tk_IGUAL tk_A tk_DNIVAL:d6
{:
      RESULT = new String(d6);
:}
            | tk_CUALES tk_SON tk_LOS tk_APELLIDOS tk_DE tk_LA tk_PERSONA
```

```
tk_CUYO tk_DNI tk_ES tk_DNIVAL:d7
{:
      RESULT = new String(d7);
:}
            | tk_CUALES tk_SON tk_LOS tk_APELLIDOS tk_DE tk_LA tk_PERSONA
tk_CUYO tk_DNI tk_ES tk_IGUAL tk_A tk_DNIVAL:d8
{:
      RESULT = new String(d8);
:}
;

PP_25 ::= PP_25_1:n tk_QST
{:
      NQLParser.resultSQL.clearResult();
      String[] nom_ap1 = n.split(":");
      NQLParser.resultSQL.getResult().append("select (case count(*) when 0
then \'no\' else \'si\' end) "+
      "as \'Respuesta\' from Persona where Nombre=\'" + nom_ap1[0] + "\' and
Apellido1=\'" + nom_ap1[1] + "\'");
:}
;
PP_25_1 ::= tk_HAY tk_ALGUIEN tk_QUE tk_SE tk_LLAME tk_NOMVAL:n1 tk_NOMVAL:a1
{:
      RESULT = new String(n1+":"+a1);
:}
            | tk_HAY tk_ALGUNA tk_PERSONA tk_QUE tk_SE tk_LLAME tk_NOMVAL:n2
tk_NOMVAL:a2
{:
      RESULT = new String(n2+":"+a2);
:}
;

PP_26 ::= PP_26_1:n tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select (case count(*) when 0
then \'no\' else \'si\' end) "+
      "as \'Respuesta\' from Persona where Nombre=\'"+n+"\'");
:}
;
PP_26_1 ::= tk_HAY tk_ALGUIEN tk_CON tk_NOMBRE tk_NOMVAL:n1
{:
      RESULT = new String(n1);
:}
            | tk_HAY tk_ALGUIEN tk_CUYO tk_NOMBRE tk_ES tk_NOMVAL:n2
{:
      RESULT = new String(n2);
:}
            | tk_HAY tk_ALGUIEN tk_QUE tk_DE tk_NOMBRE tk_SE tk_LLAME
tk_NOMVAL:n3
{:
      RESULT = new String(n3);
:}
            | tk_HAY tk_ALGUIEN tk_QUE tk_SE tk_LLAME tk_NOMVAL:n4 tk_DE
tk_NOMBRE
{:
      RESULT = new String(n4);
:}
            | tk_HAY tk_ALGUIEN tk_QUE tk_SE tk_LLAME tk_NOMVAL:n5
```

```
{:
      RESULT = new String(n5);
:}
            | tk_HAY tk_ALGUNA tk_PERSONA tk_CON tk_NOMBRE tk_NOMVAL:n6
{:
      RESULT = new String(n6);
:}
            | tk_HAY tk_ALGUNA tk_PERSONA tk_CUYO tk_NOMBRE tk_ES tk_NOMVAL:n7
{:
      RESULT = new String(n7);
:}
            | tk_HAY tk_ALGUNA tk_PERSONA tk_QUE tk_SE tk_LLAME tk_NOMVAL:n8
{:
      RESULT = new String(n8);
:}
            | tk_HAY tk_ALGUNA tk_PERSONA tk_QUE tk_SE tk_LLAME tk_NOMVAL:n9
tk_DE tk_NOMBRE
{:
      RESULT = new String(n9);
:}
;

PE_1 ::= PE_1_1 tk_DOT
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select * from Evento");
:}
;
PE_1_1 ::= tk_LISTA tk_TODOS tk_LOS tk_EVENTOS
            | tk_MUESTRAME tk_TODOS tk_LOS tk_EVENTOS
            | tk_QUIERO tk_VER tk_TODOS tk_LOS tk_EVENTOS
            | tk_LISTA tk_TODAS tk_LAS tk_REUNIONES
            | tk_MUESTRAME tk_TODAS tk_LAS tk_REUNIONES
            | tk_QUIERO tk_VER tk_TODAS tk_LAS tk_REUNIONES
;

PE_2 ::= PE_2_1:t tk_QST
{:
      NQLParser.resultSQL.clearResult();
      if (t.length() > 0) {
            String tema = new String(t.substring(0, 1).toUpperCase() +
t.substring(1));
            NQLParser.resultSQL.getResult().append("select min(Hora_inicio)
as \'Hora Inicio\' from Evento "+
            "where Fecha = (select min(Fecha) from Evento where Tema=\'" + tema
+ "\' and Fecha >= Date(Now()))");
      }
:}
;
PE_2_1 ::= tk_A tk_QUE tk_HORA tk_COMIENZA tk_LA tk_PROXIMA tk_REUNION tk_SOBRE
tk_NOMVAL:t1
{:
      RESULT = new String(t1);
:}
            | tk_A tk_QUE tk_HORA tk_EMPIEZA tk_LA tk_PROXIMA tk_REUNION
tk_SOBRE tk_NOMVAL:t2
{:
      RESULT = new String(t2);
:}
```

```
            | tk_A tk_QUE tk_HORA tk_COMENZARA tk_LA tk_PROXIMA tk_REUNION
tk_SOBRE tk_NOMVAL:t3
{:
      RESULT = new String(t3);
:}
            | tk_A tk_QUE tk_HORA tk_EMPEZARA tk_LA tk_PROXIMA tk_REUNION
tk_SOBRE tk_NOMVAL:t4
{:
      RESULT = new String(t4);
:}
            | tk_A tk_QUE tk_HORA tk_COMIENZA tk_LA tk_REUNION tk_SOBRE
tk_NOMVAL:t5
{:
      RESULT = new String(t5);
:}
            | tk_A tk_QUE tk_HORA tk_EMPIEZA tk_LA tk_REUNION tk_SOBRE
tk_NOMVAL:t6
{:
      RESULT = new String(t6);
:}
            | tk_A tk_QUE tk_HORA tk_COMENZARA tk_LA tk_REUNION tk_SOBRE
tk_NOMVAL:t7
{:
      RESULT = new String(t7);
:}
            | tk_A tk_QUE tk_HORA tk_EMPEZARA tk_LA tk_REUNION tk_SOBRE
tk_NOMVAL:t8
{:
      RESULT = new String(t8);
:}
;

PE_3 ::= PE_3_1:t tk_QST
{:
      NQLParser.resultSQL.clearResult();
      if (t.length() > 0) {
            String tema = new String(t.substring(0, 1).toUpperCase() +
t.substring(1));
            NQLParser.resultSQL.getResult().append("select distinct Hora_final
as \'Hora Final\' from Evento "+
            "where Fecha = (select min(Fecha) from Evento where Tema = \'" +
tema + "\' and Fecha >= Date(Now())) and "+
            "Hora_inicio = (select min(Hora_inicio) from Evento where Fecha =
(select min(Fecha) from Evento "+
            "where Tema = \'" + tema + "\' and Fecha >= Date(Now()))");
      }
:}
;
PE_3_1 ::= tk_A tk_QUE tk_HORA tk_ACABA tk_LA tk_PROXIMA tk_REUNION tk_SOBRE
tk_NOMVAL:t1
{:
      RESULT = new String(t1);
:}
            | tk_A tk_QUE tk_HORA tk_ACABARA tk_LA tk_PROXIMA tk_REUNION
tk_SOBRE tk_NOMVAL:t2
{:
      RESULT = new String(t2);
:}
            | tk_A tk_QUE tk_HORA tk_FINALIZARA tk_LA tk_PROXIMA tk_REUNION
```

```
tk_SOBRE tk_NOMVAL:t3
{:
      RESULT = new String(t3);
:}
            | tk_A tk_QUE tk_HORA tk_ACABA tk_LA tk_REUNION tk_SOBRE
tk_NOMVAL:t4
{:
      RESULT = new String(t4);
:}
            | tk_A tk_QUE tk_HORA tk_ACABARA tk_LA tk_REUNION tk_SOBRE
tk_NOMVAL:t5
{:
      RESULT = new String(t5);
:}
            | tk_A tk_QUE tk_HORA tk_FINALIZARA tk_LA tk_REUNION tk_SOBRE
tk_NOMVAL:t6
{:
      RESULT = new String(t6);
:}
;

PE_4 ::= PE_4_1:f tk_QST
{:
      NQLParser.resultSQL.clearResult();
      String[] ddmmyyyy = f.split("/");
      String fecha = new String(ddmmyyyy[2]+"-"+ddmmyyyy[1]+"-"+ddmmyyyy[0]);
      NQLParser.resultSQL.getResult().append("select Hora_inicio as \'Hora
Inicio\', Tema from Evento "+
      "where Fecha = \'" + fecha + "\' and Hora_inicio = (select
min(Hora_inicio) from Evento "+
      "where Fecha = \'" + fecha + "\')");
:}
;
PE_4_1 ::= tk_A tk_QUE tk_HORA tk_COMIENZA tk_LA tk_PROXIMA tk_REUNION tk_DEL
tk_DIA tk_FECHA:f1
{:
      RESULT = new String(f1);
:}
            | tk_A tk_QUE tk_HORA tk_EMPIEZA tk_LA tk_PROXIMA tk_REUNION tk_DEL
tk_DIA tk_FECHA:f2
{:
      RESULT = new String(f2);
:}
            | tk_A tk_QUE tk_HORA tk_COMENZARA tk_LA tk_PROXIMA tk_REUNION
tk_DEL tk_DIA tk_FECHA:f3
{:
      RESULT = new String(f3);
:}
            | tk_A tk_QUE tk_HORA tk_EMPEZARA tk_LA tk_PROXIMA tk_REUNION tk_DEL
tk_DIA tk_FECHA:f4
{:
      RESULT = new String(f4);
:}
            | tk_A tk_QUE tk_HORA tk_COMIENZA tk_LA tk_REUNION tk_DEL tk_DIA
tk_FECHA:f5
{:
      RESULT = new String(f5);
:}
            | tk_A tk_QUE tk_HORA tk_EMPIEZA tk_LA tk_REUNION tk_DEL tk_DIA
```

```
tk_FECHA:f6
{:
      RESULT = new String(f6);
:}
            | tk_A tk_QUE tk_HORA tk_COMENZARA tk_LA tk_REUNION tk_DEL tk_DIA
tk_FECHA:f7
{:
      RESULT = new String(f7);
:}
            | tk_A tk_QUE tk_HORA tk_EMPEZARA tk_LA tk_REUNION tk_DEL tk_DIA
tk_FECHA:f8
{:
      RESULT = new String(f8);
:}
;

PE_5 ::= PE_5_1:f tk_QST
{:
      NQLParser.resultSQL.clearResult();
      String[] ddmmyyyy = f.split("/");
      String fecha = new String(ddmmyyyy[2]+"-"+ddmmyyyy[1]+"-"+ddmmyyyy[0]);
      NQLParser.resultSQL.getResult().append("select Hora_final as \'Hora
Final\', Tema from "+
      "Evento where Fecha = \'" + fecha + "\' and Hora_inicio = (select
min(Hora_inicio) from "+
      "Evento where Fecha = \'" + fecha + "\')");
:}
;
PE_5_1 ::= tk_A tk_QUE tk_HORA tk_ACABA tk_LA tk_PROXIMA tk_REUNION tk_DEL
tk_DIA tk_FECHA:f1
{:
      RESULT = new String(f1);
:}
            | tk_A tk_QUE tk_HORA tk_ACABARA tk_LA tk_PROXIMA tk_REUNION tk_DEL
tk_DIA tk_FECHA:f2
{:
      RESULT = new String(f2);
:}
            | tk_A tk_QUE tk_HORA tk_FINALIZARA tk_LA tk_PROXIMA tk_REUNION
tk_DEL tk_DIA tk_FECHA:f3
{:
      RESULT = new String(f3);
:}
            | tk_A tk_QUE tk_HORA tk_ACABA tk_LA tk_REUNION tk_DEL tk_DIA
tk_FECHA:f4
{:
      RESULT = new String(f4);
:}
            | tk_A tk_QUE tk_HORA tk_ACABARA tk_LA tk_REUNION tk_DEL tk_DIA
tk_FECHA:f5
{:
      RESULT = new String(f5);
:}
            | tk_A tk_QUE tk_HORA tk_FINALIZARA tk_LA tk_REUNION tk_DEL tk_DIA
tk_FECHA:f6
{:
      RESULT = new String(f6);
:}
;
```

```
PE_6 ::= PE_6_1:t tk_QST
{:
      NQLParser.resultSQL.clearResult();
      if (t.length() > 0) {
            String tema = new String(t.substring(0, 1).toUpperCase() +
t.substring(1));
            NQLParser.resultSQL.getResult().append("select distinct Hora_final
as \'Hora Final\' from "+
            "Evento where Fecha = (select max(Fecha) from Evento where Tema
= \'" + tema + "\' and "+
            "Fecha <= Date(Now())) and Hora_inicio = (select max(Hora_inicio)
from Evento where "+
            "Fecha = (select max(Fecha) from Evento where Tema = \'" + tema +
"\' and "+
            "Fecha <= Date(Now()))");
      }
:}
;
PE_6_1 ::= tk_A tk_QUE tk_HORA tk_TERMINO tk_LA tk_PASADA tk_REUNION tk_SOBRE
tk_NOMVAL:t1
{:
      RESULT = new String(t1);
:}
            |  tk_A tk_QUE tk_HORA tk_FINALIZO tk_LA tk_PASADA tk_REUNION
tk_SOBRE tk_NOMVAL:t2
{:
      RESULT = new String(t2);
:}
            | tk_A tk_QUE tk_HORA tk_ACABO tk_LA tk_PASADA tk_REUNION tk_SOBRE
tk_NOMVAL:t3
{:
      RESULT = new String(t3);
:}
            | tk_A tk_QUE tk_HORA tk_TERMINO tk_LA tk_ULTIMA tk_REUNION tk_SOBRE
tk_NOMVAL:t4
{:
      RESULT = new String(t4);
:}
            | tk_A tk_QUE tk_HORA tk_FINALIZO tk_LA tk_ULTIMA tk_REUNION
tk_SOBRE tk_NOMVAL:t5
{:
      RESULT = new String(t5);
:}
            | tk_A tk_QUE tk_HORA tk_ACABO tk_LA tk_ULTIMA tk_REUNION tk_SOBRE
tk_NOMVAL:t6
{:
      RESULT = new String(t6);
:}
            | tk_A tk_QUE tk_HORA tk_TERMINO tk_LA tk_REUNION tk_SOBRE
tk_NOMVAL:t7
{:
      RESULT = new String(t7);
:}
            | tk_A tk_QUE tk_HORA tk_FINALIZO tk_LA tk_REUNION tk_SOBRE
tk_NOMVAL:t8
{:
      RESULT = new String(t8);
:}
```

```
            | tk_A tk_QUE tk_HORA tk_ACABO tk_LA tk_REUNION tk_SOBRE
tk_NOMVAL:t9
{:
      RESULT = new String(t9);
:}
;

PE_7 ::= PE_7_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select date_format(Fecha, '%d/%m/
%Y') as \'Fecha\', Hora_inicio as \'Hora Inicio\', Tema from "+
      "Evento where Fecha = (select max(Fecha) from Evento where Fecha <=
Date(Now())) and "+
      "Hora_inicio = (select max(Hora_inicio) from Evento where Fecha = (select
max(Fecha) from "+
      "Evento where Fecha <= Date(Now())))");
:}
;
PE_7_1 ::= tk_CUANDO tk_FUE tk_LA tk_ULTIMA tk_REUNION
;

PE_8 ::= PE_8_1:t tk_QST
{:
      NQLParser.resultSQL.clearResult();
      if (t.length() > 0) {
            String tema = new String(t.substring(0, 1).toUpperCase() +
t.substring(1));
            NQLParser.resultSQL.getResult().append("select date_format(Fecha,
'%d/%m/%Y') as \'Fecha\', Hora_inicio as \'Hora Inicio\' from "+
            "Evento where Tema = \'" + tema + "\' and Fecha = (select max(Fecha)
from Evento where "+
            "Fecha <= Date(Now())) and Hora_inicio = (select max(Hora_inicio)
from Evento where "+
            "Fecha = (select max(Fecha) from Evento where Fecha <=
Date(Now())))");
      }
:}
;
PE_8_1 ::= tk_CUANDO tk_FUE tk_LA tk_ULTIMA tk_REUNION tk_SOBRE tk_NOMVAL:t1
{:
      RESULT = new String(t1);
:}
;

PE_9 ::= PE_9_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select date_format(Fecha, '%d/%m/
%Y') as \'Fecha\', Hora_inicio as \'Hora Inicio\', Tema from "+
      "Evento where Fecha = (select min(Fecha) from Evento where Fecha >=
Date(Now())) and "+
      "Hora_inicio = (select min(Hora_inicio) from Evento where Fecha = (select
min(Fecha) from "+
      "Evento where Fecha >= Date(Now())))");
:}
;
PE_9_1 ::= tk_CUANDO tk_ES tk_LA tk_PROXIMA tk_REUNION
;
```

```
PE_10 ::= PE_10_1:t tk_QST
{:
      NQLParser.resultSQL.clearResult();
      if (t.length() > 0) {
            String tema = new String(t.substring(0, 1).toUpperCase() +
t.substring(1));
            NQLParser.resultSQL.getResult().append("select date_format(Fecha,
'%d/%m/%Y') as \'Fecha\', Hora_inicio as \'Hora Inicio\' from "+
            "Evento where Tema = \'" + tema + "\' and Fecha = (select min(Fecha)
from Evento where "+
            "Fecha >= Date(Now())) and Hora_inicio = (select min(Hora_inicio)
from Evento where "+
            "Fecha = (select min(Fecha) from Evento where Fecha >=
Date(Now()))))");
      }
:}
;
PE_10_1 ::= tk_CUANDO tk_ES tk_LA tk_PROXIMA tk_REUNION tk_SOBRE tk_NOMVAL:t1
{:
      RESULT = new String(t1);
:}
;

PE_11 ::= PE_11_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select count(*) from Evento where
Fecha between "+
      "(select date_add(curdate(), interval(9 - if(dayofweek(curdate())=1, 8,
dayofweek(curdate()))) day)) and "+
      "(select date_add(curdate(), interval(13 - if(dayofweek(curdate())=1, 8,
dayofweek(curdate()))) day))");
:}
;
PE_11_1 ::= tk_CUANTAS tk_REUNIONES tk_HAY tk_LA tk_SEMANA tk_QUE tk_VIENE
;

PE_12 ::= PE_12_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select * from Evento where Fecha
between "+
      "(select date_add(curdate(), interval(9 - if(dayofweek(curdate())=1, 8,
dayofweek(curdate()))) day)) and "+
      "(select date_add(curdate(), interval(13 - if(dayofweek(curdate())=1, 8,
dayofweek(curdate()))) day))");
:}
;
PE_12_1 ::= tk_QUE tk_REUNIONES tk_HAY tk_LA tk_SEMANA tk_QUE tk_VIENE
;

PE_13 ::= PE_13_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select * from Evento where "+
      "Fecha=(select date_add(curdate(), interval(9 - if(dayofweek(curdate())=1,
8, dayofweek(curdate()))) day))");
:}
```

```
;
PE_13_1 ::= tk_CUALES tk_SON tk_LAS tk_REUNIONES tk_DEL tk_LUNES tk_DE tk_LA
tk_SEMANA tk_QUE tk_VIENE
;

PE_14 ::= PE_14_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select * from Evento where "+
      "Fecha=(select date_add(curdate(), interval(10 -
if(dayofweek(curdate())=1, 8, dayofweek(curdate()))) day))");
:}
;
PE_14_1 ::= tk_CUALES tk_SON tk_LAS tk_REUNIONES tk_DEL tk_MARTES tk_DE tk_LA
tk_SEMANA tk_QUE tk_VIENE
;

PE_15 ::= PE_15_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select * from Evento where "+
      "Fecha=(select date_add(curdate(), interval(11 -
if(dayofweek(curdate())=1, 8, dayofweek(curdate()))) day))");
:}
;
PE_15_1 ::= tk_CUALES tk_SON tk_LAS tk_REUNIONES tk_DEL tk_MIERCOLES tk_DE tk_LA
tk_SEMANA tk_QUE tk_VIENE
;

PE_16 ::= PE_16_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select * from Evento where "+
      "Fecha=(select date_add(curdate(), interval(12 -
if(dayofweek(curdate())=1, 8, dayofweek(curdate()))) day))");
:}
;
PE_16_1 ::= tk_CUALES tk_SON tk_LAS tk_REUNIONES tk_DEL tk_JUEVES tk_DE tk_LA
tk_SEMANA tk_QUE tk_VIENE
;

PE_17 ::= PE_17_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select * from Evento where "+
      "Fecha=(select date_add(curdate(), interval(13 -
if(dayofweek(curdate())=1, 8, dayofweek(curdate()))) day))");
:}
;
PE_17_1 ::= tk_CUALES tk_SON tk_LAS tk_REUNIONES tk_DEL tk_VIERNES tk_DE tk_LA
tk_SEMANA tk_QUE tk_VIENE
;

PE_18 ::= PE_18_1:t tk_QST
{:
      NQLParser.resultSQL.clearResult();
      if (t.length() > 0) {
            String tema = new String(t.substring(0, 1).toUpperCase() +
t.substring(1));
```

```
            NQLParser.resultSQL.getResult().append("select Date((select Fecha
from Evento where "+
            "Tema = \'" + tema + "\' and Fecha = (select min(Fecha) from Evento
where "+
            "Fecha >= Date(Now())) and Hora_inicio = (select min(Hora_inicio)
from Evento where "+
            "Fecha = (select min(Fecha) from Evento where Fecha >=
Date(Now()))))) - Date(Now()) as \'Días\'");
}
:}
;
PE_18_1 ::= tk_CUANTOS tk_DIAS tk_FALTAN tk_PARA tk_LA tk_PROXIMA tk_REUNION
tk_SOBRE tk_NOMVAL:t1
{:
      RESULT = new String(t1);
:}
;

PE_19 ::= PE_19_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select Date((select Fecha from
Evento where "+
      "Fecha = (select min(Fecha) from Evento where Fecha >= Date(Now())) and "+
      "Hora_inicio = (select min(Hora_inicio) from Evento where Fecha = (select
min(Fecha) from "+
      "Evento where Fecha >= Date(Now()))))) - Date(Now()) as \'Días\'");
:}
;
PE_19_1 ::= tk_CUANTOS tk_DIAS tk_FALTAN tk_PARA tk_LA tk_PROXIMA tk_REUNION
;

PE_20 ::= PE_20_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select (case count(*) when 0
then \'no\' else \'si\' end) as "+
      "\'Respuesta\' from Evento where Fecha between "+
      "(select date_add(curdate(), interval(9 - if(dayofweek(curdate())=1, 8,
dayofweek(curdate()))) day)) and "+
      "(select date_add(curdate(), interval(13 - if(dayofweek(curdate())=1, 8,
dayofweek(curdate()))) day))");
:}
;
PE_20_1 ::= tk_HAY tk_ALGUNA tk_REUNION tk_LA tk_SEMANA tk_QUE tk_VIENE
;

PA_1 ::= PA_1_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select (case count(*) when 0
then \'no\' else \'si\' end) "+
      "as \'Respuesta\' from Evento inner join Asiste on Asiste.CodigoFK =
Evento.Codigo inner join "+
      "Persona on Asiste.DNIFK=Persona.DNI where Confirmado = \'1\' and Fecha =
date_add(curdate(), interval 1 day) "+
      "and Hora_inicio < Time(\'12:00:00\') and DNI=\'" + NQLParser.whoami +
"\'");
:}
```

```
;
PA_1_1 ::= tk_TENGO tk_ALGUNA tk_REUNION tk_MANANA tk_POR tk_LA tk_MANANA
;

PA_2 ::= PA_2_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select (case count(*) when 0
then \'no\' else \'si\' end) "+
      "as \'Respuesta\' from Evento inner join Asiste on Asiste.CodigoFK =
Evento.Codigo inner join Persona on "+
      "Asiste.DNIFK=Persona.DNI where Confirmado = \'1\' and Fecha =
date_add(curdate(), interval 1 day) and "+
      "DNI=\'" + NQLParser.whoami + "\'");
:}
;
PA_2_1 ::= tk_TENGO tk_ALGUNA tk_REUNION tk_MANANA
;

PA_3 ::= PA_3_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select Hora_inicio as \'Hora
inicio\', Hora_final as "+
      "\'Hora final\', Tema, Confirmado from Evento inner join Asiste on
Asiste.CodigoFK = Evento.Codigo "+
      "inner join Persona on Asiste.DNIFK=Persona.DNI where Fecha =
date_add(curdate(), interval 1 day) and "+
      "Hora_inicio < Time(\'12:00:00\') and DNI=\'" + NQLParser.whoami + "\'");
:}
;
PA_3_1 ::= tk_QUE tk_REUNIONES tk_TENGO tk_MANANA tk_POR tk_LA tk_MANANA
;

PA_4 ::= PA_4_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select Hora_inicio as \'Hora
inicio\', Hora_final as "+
      "\'Hora final\', Tema, Confirmado from Evento inner join Asiste on
Asiste.CodigoFK = Evento.Codigo "+
      "inner join Persona on Asiste.DNIFK=Persona.DNI where Fecha =
date_add(curdate(), interval 1 day) and "+
      "DNI=\'" + NQLParser.whoami + "\'");
:}
;
PA_4_1 ::= tk_QUE tk_REUNIONES tk_TENGO tk_MANANA
;

PA_5 ::= PA_5_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select count(*) as \'Personas\',
Hora_inicio as \'Hora inicio\', "+
      "Tema from Evento inner join Asiste on Asiste.CodigoFK = Evento.Codigo
where Confirmado = \'1\' and "+
      "Fecha = curdate() group by Codigo");
:}
;
```

```
PA_5_1 ::= tk_CUANTAS tk_PERSONAS tk_HAN tk_CONFIRMADO tk_SU tk_ASISTENCIA tk_A
tk_LA tk_REUNION tk_DE tk_HOY
;

PA_6 ::= PA_6_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select count(*) as \'Personas\',
Hora_inicio as \'Hora inicio\', "+
      "Tema from Evento inner join Asiste on Asiste.CodigoFK = Evento.Codigo
where Confirmado = \'1\' and "+
      "Fecha = date_add(curdate(), interval 1 day) group by Codigo");
:}
;
PA_6_1 ::= tk_CUANTAS tk_PERSONAS tk_HAN tk_CONFIRMADO tk_SU tk_ASISTENCIA tk_A
tk_LA tk_REUNION tk_DE tk_MANANA
;

PA_7 ::= PA_7_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select count(*) as \'Personas\',
Hora_inicio as \'Hora inicio\', "+
      "Tema from Evento inner join Asiste on Asiste.CodigoFK = Evento.Codigo
where Confirmado = \'1\' and "+
      "Fecha = date_sub(curdate(), interval 1 day) group by Codigo");
:}
;
PA_7_1 ::= tk_CUANTAS tk_PERSONAS tk_CONFIRMARON tk_SU tk_ASISTENCIA tk_A tk_LA
tk_REUNION tk_DE tk_AYER
;

PA_8 ::= PA_8_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select count(*) as \'Personas\',
date_format(Fecha, \'%d/%m/%Y\') as "+
      "\'Fecha\', Hora_inicio as \'Hora inicio\', Tema from Evento inner join
Asiste on Asiste.CodigoFK = Evento.Codigo "+
      "where Confirmado = \'1\' and Fecha = (select date_add(curdate(),
interval(9 - if(dayofweek(curdate())=1, 8, dayofweek(curdate())))) day)) "+
      "group by Codigo");
:}
;
PA_8_1 ::= tk_CUANTAS tk_PERSONAS tk_HAN tk_CONFIRMADO tk_SU tk_ASISTENCIA tk_A
tk_LA tk_REUNION tk_DEL tk_LUNES tk_DE tk_LA tk_SEMANA tk_QUE tk_VIENE
;

PA_9 ::= PA_9_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select count(*) as \'Personas\',
date_format(Fecha, \'%d/%m/%Y\') as "+
      "\'Fecha\', Hora_inicio as \'Hora inicio\', Tema from Evento inner join
Asiste on Asiste.CodigoFK = Evento.Codigo "+
      "where Confirmado = \'1\' and Fecha = (select date_add(curdate(),
interval(10 - if(dayofweek(curdate())=1, 8, dayofweek(curdate())))) day)) "+
      "group by Codigo");
:}
```

```
;
PA_9_1 ::= tk_CUANTAS tk_PERSONAS tk_HAN tk_CONFIRMADO tk_SU tk_ASISTENCIA tk_A
tk_LA tk_REUNION tk_DEL tk_MARTES tk_DE tk_LA tk_SEMANA tk_QUE tk_VIENE
;

PA_10 ::= PA_10_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select count(*) as \'Personas\',
date_format(Fecha, \'%d/%m/%Y\') as "+
      "\'Fecha\', Hora_inicio as \'Hora inicio\', Tema from Evento inner join
Asiste on Asiste.CodigoFK = Evento.Codigo "+
      "where Confirmado = \'1\' and Fecha = (select date_add(curdate(),
interval(11 - if(dayofweek(curdate())=1, 8, dayofweek(curdate()))) day)) "+
      "group by Codigo");
:}
;
PA_10_1 ::= tk_CUANTAS tk_PERSONAS tk_HAN tk_CONFIRMADO tk_SU tk_ASISTENCIA tk_A
tk_LA tk_REUNION tk_DEL tk_MIERCOLES tk_DE tk_LA tk_SEMANA tk_QUE tk_VIENE
;

PA_11 ::= PA_11_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select count(*) as \'Personas\',
date_format(Fecha, \'%d/%m/%Y\') as "+
      "\'Fecha\', Hora_inicio as \'Hora inicio\', Tema from Evento inner join
Asiste on Asiste.CodigoFK = Evento.Codigo "+
      "where Confirmado = \'1\' and Fecha = (select date_add(curdate(),
interval(12 - if(dayofweek(curdate())=1, 8, dayofweek(curdate()))) day)) "+
      "group by Codigo");
:}
;
PA_11_1 ::= tk_CUANTAS tk_PERSONAS tk_HAN tk_CONFIRMADO tk_SU tk_ASISTENCIA tk_A
tk_LA tk_REUNION tk_DEL tk_JUEVES tk_DE tk_LA tk_SEMANA tk_QUE tk_VIENE
;

PA_12 ::= PA_12_1 tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select count(*) as \'Personas\',
date_format(Fecha, \'%d/%m/%Y\') as "+
      "\'Fecha\', Hora_inicio as \'Hora inicio\', Tema from Evento inner join
Asiste on Asiste.CodigoFK = Evento.Codigo where "+
      "Confirmado = \'1\' and Fecha = (select date_add(curdate(), interval(13 -
if(dayofweek(curdate())=1, 8, dayofweek(curdate()))) day)) "+
      "group by Codigo");
:}
;
PA_12_1 ::= tk_CUANTAS tk_PERSONAS tk_HAN tk_CONFIRMADO tk_SU tk_ASISTENCIA tk_A
tk_LA tk_REUNION tk_DEL tk_VIERNES tk_DE tk_LA tk_SEMANA tk_QUE tk_VIENE
;

PA_13 ::= PA_13_1:n tk_QST
{:
      NQLParser.resultSQL.clearResult();
      NQLParser.resultSQL.getResult().append("select datediff((select min(Fecha)
from Evento inner join Asiste on "+
      "Asiste.CodigoFK=Evento.Codigo inner join Persona on
```

```
Asiste.DNIFK=Persona.DNI where "+
      "Persona.Nombre = \'" + n + "\' and Fecha > curdate()), curdate())
as \'Días\'");
:}
;
PA_13_1 ::= tk_CUANTOS tk_DIAS tk_FALTAN tk_PARA tk_LA tk_PROXIMA tk_REUNION
tk_CON tk_NOMVAL:n1
{:
      RESULT = new String(n1);
:}
;

PA_14 ::= PA_14_1:t tk_QST
{:
      NQLParser.resultSQL.clearResult();
      if (t.length() > 0) {
            String tema = new String(t.substring(0, 1).toUpperCase() +
t.substring(1));
      NQLParser.resultSQL.getResult().append("select count(distinct Codigo)
as \'Reuniones\' from Evento inner join Asiste on "+
      "Asiste.CodigoFK=Evento.Codigo where month(Fecha) = month(now()) and Fecha
> curdate() and "+
      "Tema = \'" + tema + "\'");
}
:}
;
PA_14_1 ::= tk_CUANTAS tk_REUNIONES tk_SOBRE tk_NOMVAL:t1 tk_HAREMOS tk_ESTE
tk_MES
{:
      RESULT = new String(t1);
:}
;

PA_15 ::= PA_15_1:t tk_QST
{:
      NQLParser.resultSQL.clearResult();
      if (t.length() > 0) {
            String tema = new String(t.substring(0, 1).toUpperCase() +
t.substring(1));
      NQLParser.resultSQL.getResult().append("select count(distinct Codigo)
as \'Reuniones\' from "+
      "Evento inner join Asiste on Asiste.CodigoFK=Evento.Codigo where
yearweek(Fecha) = yearweek(curdate()) and "+
      "Tema = \'" + tema + "\'");
}
:}
;
PA_15_1 ::= tk_CUANTAS tk_REUNIONES tk_SOBRE tk_NOMVAL:t1 tk_HAREMOS tk_ESTA
tk_SEMANA
{:
      RESULT = new String(t1);
:}
;
```