E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Sistema gestor de ensayos en amortiguadores

Grado en Ingeniería
en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Autora: Ángela Lerín Vigara

Director: Israel Arnedo Gil

Pamplona, 28 de junio 2017

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Resumen

El proyecto se ha llevado a cabo durante las prácticas curriculares de la autora en la empresa Engineea Remote Technologies.

Describe el desarrollo de un sistema compuesto por: una base de datos, una interfaz gráfica de usuario (GUI), y un dispositivo de adquisición de datos (DAQ).

Los objetivos a conseguir son:

- ✓ Partiendo de una base de datos original recibida del cliente, se diseñará una nueva base con distinta estructura y formato.
  La base de datos original contiene información sobre los amortiguadores que se producen en una fábrica y los resultados numéricos de ensayos de calidad. Se migrarán estos datos a la base de datos nueva.
  La migración se hará automáticamente ejecutando clases programadas en lenguaje Java, y con conocimientos de Lenguaje de Consulta Estructurado (SQL).

- ✓ Se creará una interfaz gráfica de usuario para consultar y hacer operaciones en la base de datos nueva. Se desarrolla en Java con la ayuda del Entorno de Desarrollo Integrado (IDE) Netbeans y diversas Interfaces de Programación de Aplicaciones (API).

- ✓ Por último, la interfaz enviará órdenes a las máquinas que ejecutan ensayos mediante el DAQ y guardará los resultados en la base. Se escribirá el código para una simulación de su comportamiento.

Palabras clave

Java, amortiguadores, bases de datos, SQL, Interfaz Gráfica de Usuario

Abstract

This project has been completed during an internship in Engineea Remote Technologies Company by the author.

This document introduces the development of a system that includes: a database, a Graphical User Interface (GUI), and a Data Adquisition device (DAQ).

Goals are:

- ✓ Given the current client database, a new one will be designed with different architecture and format.
  The original database stores information about shock absorbers that are produced in a factory. Also, it stores numeric results of quality tests done. This data will be migrated to the new database.
  The migration will be completed automatically executing Java classes and by making use of Structured Query Language, SQL.

- ✓ A Graphical User Interface will be made for reading and operating with the new database. It will be developed with the help of an Integrated Development Environment (IDE) called Netbeans, and several Application Programming Interfaces (API).

- ✓ Finally, the interface will send orders to the machines that execute the tests via DAQ and will storage results on the database. Written code will contain a simulation for this behaviour.

Keywords

Java, shock absorbers, databases, SQL, Graphic User Interface

Índice de contenidos

Agradecimientos

Me gustaría dar las gracias al equipo de Engineea que tan positivamente me ha acogido en este semestre de prácticas. Me han ofrecido una gran experiencia y han favorecido a mi desarrollo, tanto a nivel formativo, como socio-laboral.

En especial agradezco la coordinación técnica de Iñigo Sola, y la dirección de estas prácticas, por parte de Fernando Ruiz.

También quiero agradecer a los docentes en la Universidad Pública de Navarra que me han ayudado en todo este proceso con su orientación: Israel Arnedo, director del presente proyecto, y Antonio López, responsable de las prácticas curriculares.

A mi compañero y gran amigo Samuel, que siempre ha creído en mí.

1. Introducción

1.1 Contexto

Se han realizado unas prácticas curriculares en Engineea, una empresa dedicada a la innovación y a la resolución de problemas tecnológicos.

El cliente de este proyecto es otra empresa, dedicada a la fabricación de amortiguadores.

La empresa solicita actualizar sus herramientas en la gestión de ensayos de calidad debido a que demandan características con las que no cuentan; y poseen algunas obsoletas.

En definitiva, se trata de ofrecerles soluciones que se adecúen al uso normal de la fábrica en el presente.

Ilustración 1: máquina de ensayos

1.2 Descripción de la memoria

Este documento se desglosa en tres partes principales: el tratamiento de bases de datos, la interfaz gráfica de usuario y el control de un dispositivo hardware.

Dentro de cada apartado, se concretarán los objetivos y los requisitos; detallando a continuación el desarrollo para cumplirlos. Acompañados del diseño, capturas y diagramas necesarios.

Se cierra la memoria con un apartado de líneas futuras y unas conclusiones.

Dada la particularidad del objeto del proyecto y el cliente, se incluye un ANEXO A:

para comprender la temática y un listado de las tecnologías informáticas usadas.

Se adhieren anexos que conectan el plan de estudios de la universidad, con los conocimientos necesarios y adquiridos a lo largo de estas prácticas en Engineea.

2. Diseño y creación de la base de datos nueva

2.1 Objetivos

El principal objetivo es tener un almacén de datos fácilmente manejable y con el formato de archivo especificado por el cliente.

Para ello se siguen una serie de pasos:

1. Analizar la base de datos original. Para conocer lo que hay almacenado en ella.

2. Diseñar la base de datos nueva. En base al conocimiento de la base original y aportando mejoras respecto a esta.

3. Crear la base de datos nueva en el formato requerido.

4. Traspasar la base de datos original del formato actual al deseado. De este modo la transferencia de los datos entre las bases es directa.

5. Programar las clases que migran automáticamente los datos de la base original a la base nueva.

2.2 Análisis de la base de datos original

Como ayuda a la creación de la base nueva, el cliente proporciona la base actual cargada con datos.

El primer obstáculo es poder acceder a esta base: fue creada en formato *.mbd*, propio de Microsoft Access, versión 97. Este formato está obsoleto, y no es posible abrirlo desde versiones actuales de Access. Concretamente, a partir de la versión del año 2007 [1].

La solución es abrir la base desde una versión de Access 2003 en una máquina virtual, con sistema operativo Windows XP.

Gracias al software de diagramas DIA se esbozan las tablas de la base original, sus campos (sinónimo de atributos o columnas) y tipos de datos contenidos: Ilustración 2: Base de datos original .

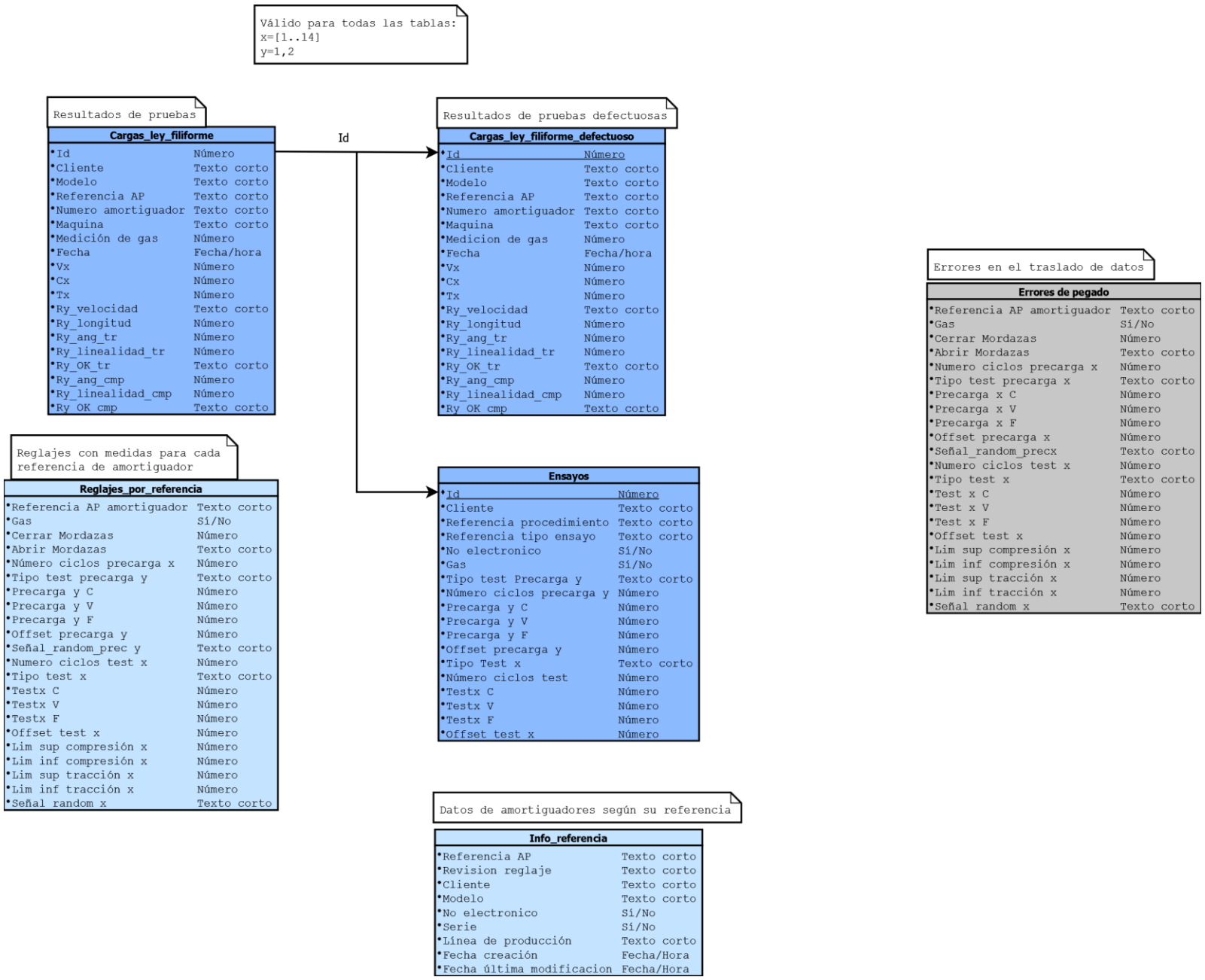


Ilustración 2: Base de datos original

Antes de comenzar se recomienda hacer una lectura del ANEXO A: Glosario de términos.

2.2.1 Arquitectura de la base de datos original

La base se compone de 5 tablas:

- **Tabla *Info_referencia***: contiene información genérica de los amortiguadores. Como el cliente de la pieza, la línea de producción, el modelo de pieza, etc.

- **Tabla *Reglajes_por_referencia***: contiene las características del ensayo que se debe realizar (reglaje) según la equivalencia (conjunto de referencias).

- **Tabla *Ensayos***: contiene pocos registros (sinónimo de filas) con algunas características generales de ensayos. Se ignorarán, puesto que son pruebas de funcionamiento del programa y la base en la fábrica.

- **Tabla *Cargas_ley_filiforme***: contiene los resultados de ensayos de calidad ejecutados.

- **Tabla *Cargas_ley_filiforme_defectuoso***: equivalente a la anterior.
  En esta tabla se guardan los resultados de aquellos amortiguadores que se han definido como defectuosos por no cumplir lo especificado en la tabla *Reglajes_por_referencia*. Dicho de otro modo, sus valores de carga medidos en los tests están fuera de los parámetros límite definidos para su reglaje.

2.2.2 Características de la base de datos original

Se comentan algunas características de la base:

- Hay tablas que albergan el mismo campo. Por ejemplo, los nombres de los amortiguadores, y su cliente; son atributos que aparecen en tres de las cinco tablas. Incluso, el primero de ellos aparece con distintos títulos: “Referencia AP amortiguador” en la tabla *Info_referencia* es equivalente a “Referencia AP” en otras tablas.

- Algunos campos al ser de tipo texto permiten introducir cualquier carácter alfanumérico. Eso resulta en datos que no se corresponden con la descripción intuitiva del atributo. Por ejemplo, existen frases escritas por operarios en el campo “Numero amortiguador” de la tabla *Cargas_ley_filiforme*, cuando lo esperado sería tener únicamente números. La mayor parte de registros tienen aquí un número entero.

- Se permiten los valores nulos (NULL) en todos los campos y tablas.

  En la tabla *Cargas_ley_filiforme* se han encontrado cuatro registros con el campo “Referencia AP” igual a NULL. Esto impide conocer de qué amortiguadores se tratan.

- Se aglutina mucha información variada en pocas tablas.
  Por ejemplo, se tienen en la tabla *Reglajes_por_referencia* los dos campos de precarga y las catorce condiciones de los ensayos; resultando en 173 campos. De los cuales muchos no contienen datos útiles (son nulos o 0).

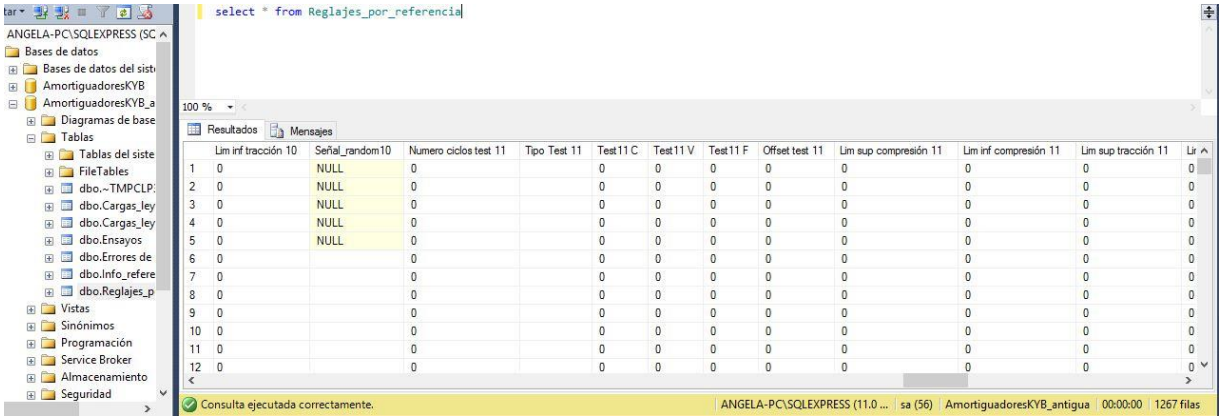


Ilustración 3: tabla Reglajes_por_referencia, y algunos de sus datos almacenados en la base de datos original

- Existen campos de identificación ("id") en algunas tablas, pero no son utilizados debidamente: no coinciden entre tablas y no hay relaciones de identidad establecidas.

- En alguna operación anterior a la recepción de la base, han debido de surgir errores, tales que el programa (Access) creó tablas de error propias: tabla *Errores de pegado*, en color gris en la Ilustración 2: Base de datos original.

2.3 Diseño de la base de datos nueva

2.3.1 Requisitos de diseño de la base de datos nueva

En principal objetivo es establecer un modelo normalizado de base de datos.

De forma breve:

El modelo normalizado se trata de un estándar cuyo fin es evitar errores comunes en el diseño de bases de datos relacionales, descrito por Edgar Frank Codd en la década de los 80 [2].

Está compuesto de una serie de formas de normalización.

Para este proyecto se ha considerado suficiente implementar la primera forma normal (1FN); puesto que supone ya una gran mejora respecto al diseño original.

Se presentan los conceptos de 1FN resumidos en el siguiente párrafo [3]:

Una tabla está normalizada en la primera forma si:

- *Todos los atributos son atómicos. Un atributo es atómico si los elementos del dominio son simples e indivisibles.*
- *La tabla contiene una clave primaria* única.*
- *La clave primaria no contiene atributos nulos.*
- *No debe existir variación en el número de columnas.*
- *Los campos no clave deben identificarse por la clave (Dependencia Funcional).*
- *Debe existir una independencia del orden tanto de las filas como de las columnas, es decir, si los datos cambian de orden no deben cambiar sus significados.*

Eliminando valores repetidos en una base de datos.

Numerando los requisitos de la base de datos nueva:

1. **Todas las tablas poseerán un campo "id".**

Una clave primaria (*) es un atributo que identifica de manera única cada registro en una tabla. Se establecerá que los campos llamados "id" sean esta clave primaria.

2. **Se implementará integridad referencial.**

Las claves externas (o foráneas), son aquellos atributos de una tabla que enlazan con una clave primaria de otra tabla. En el diseño de la nueva base de datos, se les ha asignado títulos con formato: tabla_atributo-de-tabla. Por ejemplo: "equivalencia_id", es una clave foránea en la tabla *referencia*, procede de la tabla *equivalencia* y se trata del atributo "id".

Esta restricción deriva en que sólo es posible introducir valores en claves foráneas que existan como clave primaria en la tabla donde proceden. Es decir: no podría hacerse referencia a la "id" de un cliente en la tabla *reglaje*, si dicho cliente no existe antes en la tabla *cliente*. A esto se le conoce como integridad referencial.

3. **Notación estandarizada para tablas y campos; y tipos de datos definidos**

Todas las tablas tendrán por simplicidad una única palabra como nombre, en minúsculas y en singular.

Los nombres compuestos de atributos se escriben de acuerdo a un estándar conocido como camelCase [4], por ejemplo "cargaTraccion", "numCondicion", o "limiteSupCompresion". Se hace así cuando no es posible describir el atributo con una única palabra.

Se pretende dotar de coherencia a la nueva base y controlar la inserción de datos. Cada campo estará definido por un tipo, no pudiendo introducir valores inesperados o nulos (en general, aunque hay casos excepcionales).

4. **Las tablas serán de propósito único.**

Esto es, una tabla guardará exclusivamente nombres de referencias; otra guardará nombres de clientes, etc.

Se produce un incremento entonces del número de tablas, con la ventaja de reducir el número de atributos en ellas.

Por ejemplo, se eliminará el uso de catorce campos de condiciones, empleando sólo una columna que indique el número de la condición. Esto además elimina registros sin información útil: aquellas condiciones de velocidad, frecuencia, amplitud de señal…de valor 0.

La mayor parte de reglajes recibidos del cliente, contenían ceros en la condición número 8 y superior.

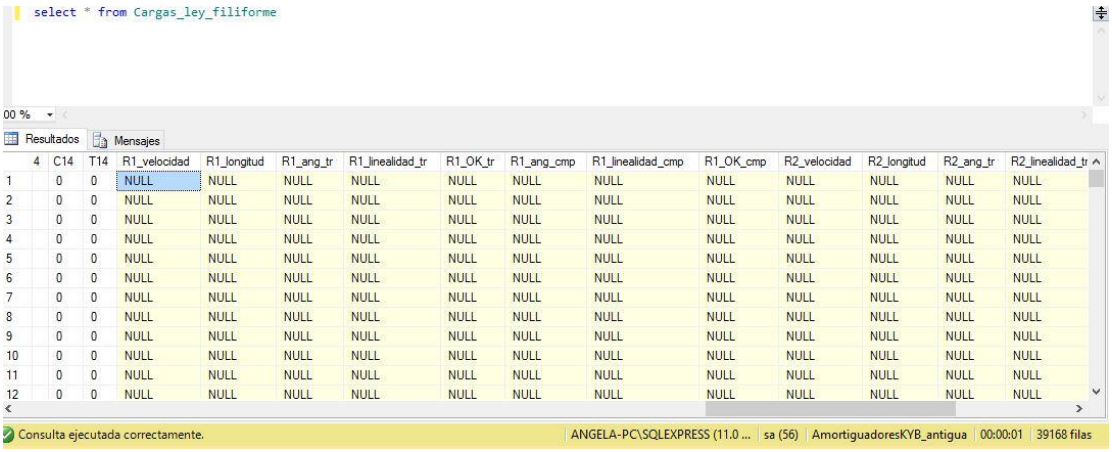
Con el nuevo diseño, se podrá añadir si se desea, un mayor número de condiciones. En la base de datos original está fija en catorce.

Igualmente ocurre con otras entidades únicas, como clientes, tipos de señal o máquina de producción; que al separar en tablas facilitan la incorporación de nuevos nombres o tipos sin cambiar la base de datos.

5. **Campos sin uso en la base original no se incluirán en la base nueva**

No se van a incorporar atributos en estas tablas que, el cliente afirma, no tienen uso o dejarán de tenerlo en el futuro.

Por ejemplo, se ignorarán los datos de los campos “abrir mordazas” y “cerrar mordazas” de la tabla *Reglajes_por_referencia,* los campos de señales aleatorias “random” en *Reglajes_por_referencia*, o con formato Rx_velocidad/ longitud/ linealidad… En la tabla *Cargas_ley_filiforme*. Que sólo contienen nulos.



	4	C14	T14	R1_velocidad	R1_longitud	R1_ang_tr	R1_linealidad_tr	R1_OK_tr	R1_ang_cmp	R1_linealidad_cmp	R1_OK_cmp	R2_velocidad	R2_longitud	R2_ang_tr	R2_linealidad_tr
1		0	0	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
2		0	0	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
3		0	0	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
4		0	0	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
5		0	0	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
6		0	0	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
7		0	0	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
8		0	0	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
9		0	0	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
10		0	0	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
11		0	0	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
12		0	0	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Ilustración 4: campos sin uso en la tabla Cargas_ley_filiforme, de la base original. Todos los datos son nulos.

6. **Petición del cliente: filtrado de resultados por referencia**

El cliente quiere poder filtrar los resultados de ensayos por referencias individuales, en contraposición con su método actual (por grupos de referencias).

Pero a la vez, no se pueden ignorar las equivalencias, ya que los reglajes se asocian a ellas y es el modo de trabajo habitual.

Por esto serán necesarias dos tablas: *equivalencia* y *referencia*.

Durante su diseño, se pensó que la tabla *referencia* debería tener un campo con el nombre de otra referencia con la que estuviese agrupada. Pero esto suponía multiplicar los registros.

Ejemplo simplificado: si una equivalencia es 1/2/3/4, las referencias en la tabla *referencia* se verían como:

Nombre	Nombre_2
1	2
1	3
1	4
2	1
2	3
2	4
3	1
. . .	. . .

Tabla 1: Solución de asignación de grupos entre referencias

Se verá con detalle en el apartado 2.3.2: Arquitectura de la base de datos nueva, que al final se decide implementar una columna con el id de la equivalencia correspondiente en la tabla *equivalencia*.

7. **Se evitará la pérdida o modificación de datos relativos a resultados**

Dado que es posible modificar los reglajes, equivalencias y referencias; tiene que existir un mecanismo que evite que los cambios tengan efecto en resultados guardados con anterioridad.

Por ejemplo: se tienen ensayos de un amortiguador hechos con un cierto reglaje. Más adelante, se modifica dicho reglaje. Pero no sería correcto que, al consultar los resultados, apareciesen los datos de reglaje nuevos; sino los originales.

Para ello, se añadirá a algunas tablas el campo “borrado”, cuyo valor (0 ó 1), indicará si el registro es válido y en uso, o no.

Aparecerá en las tablas: *referencia*, *equivalencia* o *reglaje*.

A efectos prácticos, los registros con el campo “borrado” igual a 1 no tendrán uso porque han sido “borrados”; pero pueden ser consultados desde los resultados porque siguen existiendo en la tabla.

2.3.2 Arquitectura de la base de datos nueva

Cumpliendo con los requisitos expuestos, se diseña una base compuesta por once tablas.

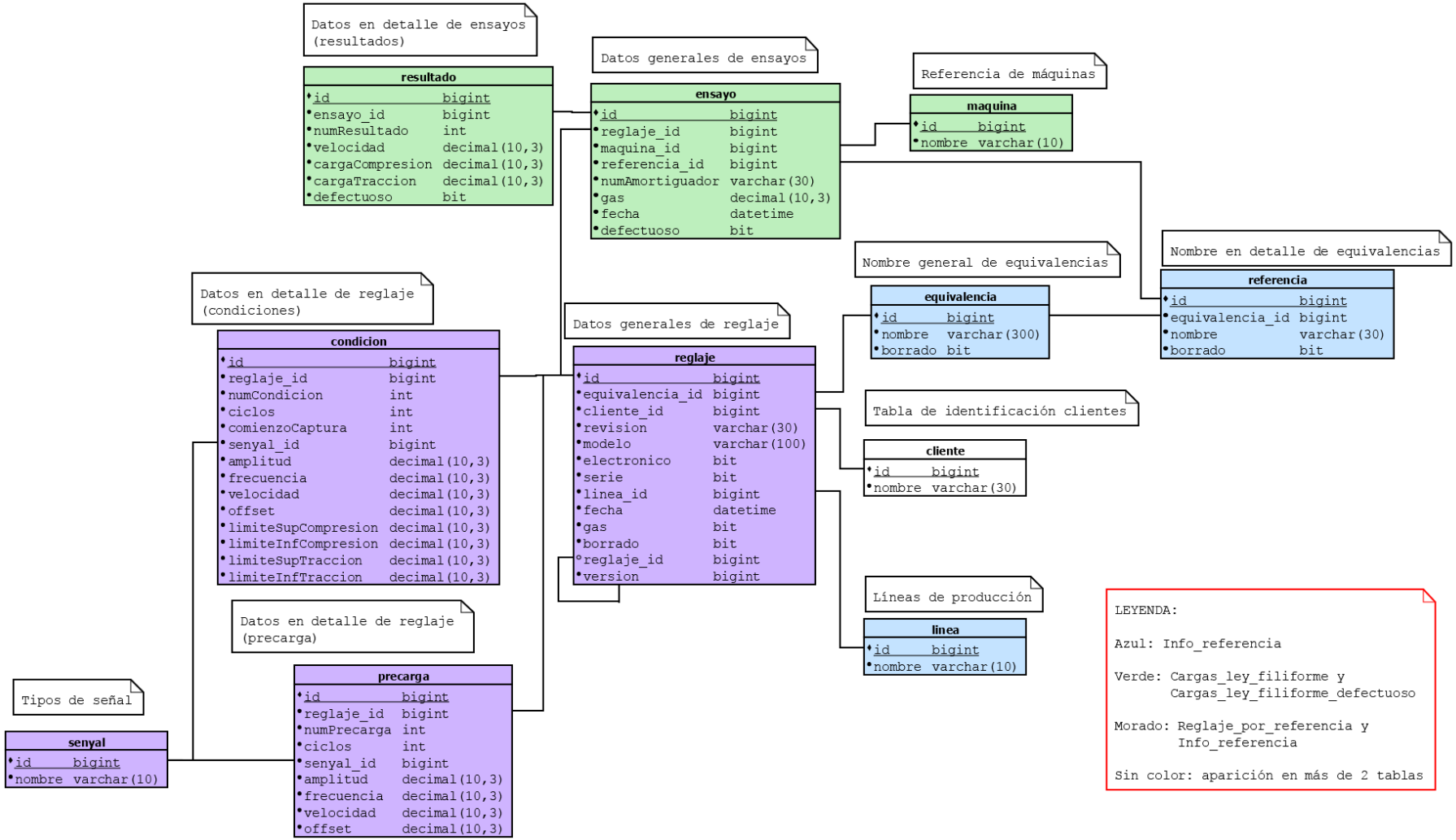


Ilustración 5: esquema de la base de datos nueva

Nota: las tablas mostradas a continuación son capturas del software SQL Server Management Studio, de gestión de bases de datos SQL Server. Contienen ya datos cargados en la base de datos nueva gracias a la migración. Se muestran así para que se comprenda la función de cada campo.

- **Tabla *senyal***: contiene los tipos de señales empleadas en los tests. Se ha evitado el uso de ñ en el nombre por compatibilidad con sistemas en otros idiomas. A veces estos introducen símbolos cuando no leen correctamente un carácter desconocido.

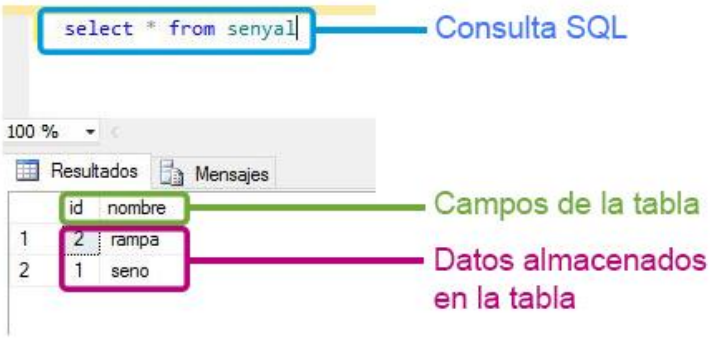
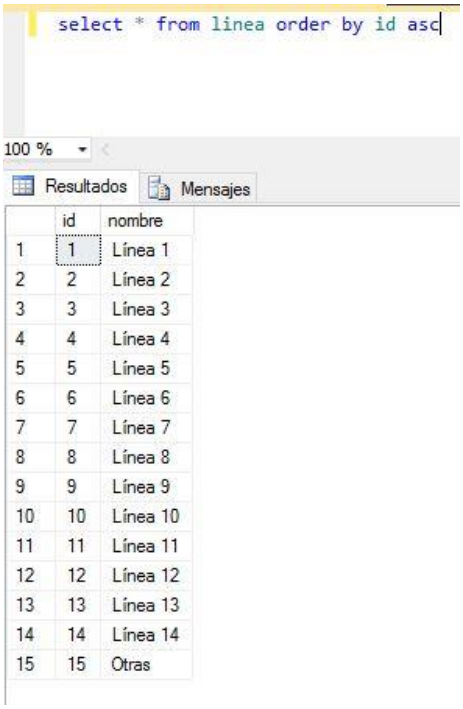


Ilustración 6: Tabla senyal, consultada desde el software de gestión de la base de datos

- **Tabla *linea***: contiene las líneas de producción de la fábrica.



	id	nombre
1	1	Línea 1
2	2	Línea 2
3	3	Línea 3
4	4	Línea 4
5	5	Línea 5
6	6	Línea 6
7	7	Línea 7
8	8	Línea 8
9	9	Línea 9
10	10	Línea 10
11	11	Línea 11
12	12	Línea 12
13	13	Línea 13
14	14	Línea 14
15	15	Otras

Ilustración 7: Tabla linea

- **Tabla *cliente:*** contiene los nombres de clientes de amortiguadores de automoción.

```
select * from cliente order by id
```

100 %

Resultados | Mensajes

	id	nombre
1	1	GENERAL MOTORS
2	2	SUZUKI
3	3	RENAULT
4	4	SEAT
5	5	MITSUBISHI
6	6	VOLKSWAGEN
7	7	INTERNO AP
8	8	PATRONES
9	9	NISSAN
10	10	CITROEN
11	11	BENTELER
12	12	TOYOTA
13	13	MAZDA

Ilustración 8: Tabla cliente

- **Tabla *referencia***: guarda las referencias de amortiguadores. Cada referencia pertenece a su vez a una equivalencia registrada en la tabla *equivalencia*.
  El puntero que relaciona ambas tablas es la clave foránea: "equivalencia_id".

```
select * from referencia order by id desc
```

100 %

Resultados | Mensajes

	id	equivalencia_id	nombre	borrado
103	2064	1167	9411	0
104	2063	1166	9407	0
105	2062	1165	9406	0
106	2061	1164	9405	0
107	2060	1163	9382	0
108	2059	1162	9956	0
109	2058	1162	9961	0
110	2057	1162	9375	0
111	2056	1162	9370	0
112	2055	1161	9960	0
113	2054	1161	9957	0
114	2053	1161	9643	0
115	2052	1161	9369	0

Ilustración 9: Tabla referencia

- **Tabla *equivalencia***: guarda las equivalencias. En ocasiones la equivalencia es sólo una referencia única, y en otras un conjunto de referencias.

```
select * from equivalencia
```

100 %

Resultados | Mensajes

	id	nombre	borrado
43	43	3874	0
44	44	3875/3878	0
45	45	3876	0
46	46	3877	0
47	47	4174	0
48	48	4339	0
49	49	4340/4341/5371/5372	0
50	50	4344/4345/5369/5370	0
51	51	4456	0
52	52	4464/5364	0
53	53	4502/4503/4575	0
54	54	4525	0
55	55	4557	0

Ilustración 10: Tabla equivalencia

- **Tabla *maquina***: contiene los nombres de las máquinas de producción de la fábrica.

```
select * from maquina
```

100 %

Resultados | Mensajes

	id	nombre
1	2	MTS10255
2	1	MTS10278
3	3	Otra

Ilustración 11: Tabla maquina

- **Tabla *reglaje***: contiene datos generales de una equivalencia. Varios de sus atributos son claves foráneas: “cliente_id” ó “senyal_id” son identificadores de registros concretos en las tablas *cliente* y *senyal*.

```
select * from reglaje
```

100 %

Resultados | Mensajes

	id	equivalencia_id	cliente_id	revision	modelo	electronico	serie	linea_id	fecha	gas	borrado	reglaje_id	version
73	73	1124	20	-	PSA PF2 B7	0	1	5	2014-05-14 00:00:00.000	1	0	73	0
74	74	741	11	-	NCV3 T5 CE6 ...	0	1	9	2006-05-10 00:00:00.000	1	0	74	0
75	75	690	14	B	CINQUECENT...	0	1	3	2006-05-26 00:00:00.000	1	0	75	0
76	76	91	3	-	X-95 DEL.	0	1	15	2009-03-17 00:00:00.000	1	0	76	0
77	77	1132	20	-	PSA PF2 T84	0	1	5	2014-11-01 00:00:00.000	1	0	77	0
78	78	1158	20	-	PF2 T75 RACING	0	1	5	2014-03-18 00:00:00.000	1	0	78	0
79	79	487	12	A	AVENSIS Del.	0	1	7	2004-08-31 00:00:00.000	1	0	79	0
80	80	270	3	1	X-76 TRS	0	1	15	2016-06-08 00:00:00.000	1	0	80	0
81	81	32	3	-	SAFRANE FRO...	0	1	3	2016-11-22 00:00:00.000	1	0	81	0
82	82	11	9	A	X-11 T	0	1	4	2007-03-06 00:00:00.000	1	0	82	0
83	83	179	3	-	X73 LIGERO Del.	0	1	9	2002-05-20 00:00:00.000	0	0	83	0
84	84	249	18	A	V184/185 DELA...	0	1	9	2002-05-20 00:00:00.000	1	0	84	0
85	85	180	19	-	A03 Trs.	0	1	8	2003-12-09 00:00:00.000	1	0	85	0

Ilustración 12: Tabla reglaje

- **Tabla *condicion***: guarda las medidas que se aplican a la máquina en los tests de calidad. Es una parte del reglaje, por lo que tiene un campo de "reglaje_id" que apunta al que corresponde.
  Se ha diseñado un campo, "numCondicion", en el que se especifique a qué número de prueba se hace mención (1-14).

```
select * from condicion
```

100 %

Resultados | Mensajes

	id	reglaje_id	numCondicion	senyal_id	amplitud	frecuencia	velocidad	offset	limiteSupCompresion	limiteInfCompresion	limiteSupTraccion	limiteInfTraccion	ciclos	comienzoCaptura
1	1	1	5	1	15.920	4.000	400.000	25.000	70.200	52.900	76.100	57.700	3	2
2	2	1	4	1	15.920	3.000	300.000	25.000	61.000	45.400	52.100	38.100	3	2
3	3	1	3	1	10.610	3.000	200.000	25.000	49.700	36.100	31.400	21.200	3	2
4	4	1	2	1	10.610	1.500	100.000	25.000	29.300	19.500	16.400	8.800	3	2
5	5	1	1	1	5.310	1.500	50.000	25.000	15.300	7.900	11.200	4.600	3	2
6	6	1	6	1	14.590	6.000	550.000	25.000	82.900	63.300	101.900	78.900	3	2
7	7	1	7	1	14.920	8.000	750.000	25.000	98.000	75.600	130.200	102.000	3	2
8	8	1	8	1	15.120	10.000	950.000	25.000	111.300	86.500	158.200	124.900	3	2
9	9	1	9	1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0	0
10	10	1	10	1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0	0
11	11	1	11	1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0	0
12	12	1	12	1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0	0
13	13	1	13	1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0	0

Ilustración 13: tablaCondicion

- **Tabla *precarga***: similar a la tabla *condicion*, pero sin los atributos de límites. También forma parte del reglaje.

```
select * from precarga
```

100 %

Resultados | Mensajes

	id	reglaje_id	numPrecarga	ciclos	senyal_id	amplitud	frecuencia	velocidad	offset
1	1	1	1	6	1	21.220	3.000	400.000	0.000
2	2	1	2	0	1	0.000	0.000	0.000	0.000
3	3	2	1	6	1	21.220	3.000	400.000	0.000
4	4	2	2	0	1	0.000	0.000	0.000	0.000
5	5	3	1	6	1	25.000	1.910	300.000	0.000
6	6	3	2	0	1	0.000	0.000	0.000	0.000
7	7	4	1	6	1	25.000	1.910	300.000	0.000
8	8	4	2	0	1	0.000	0.000	0.000	0.000
9	9	5	1	15	1	30.000	2.650	500.000	0.000
10	10	5	2	0	1	0.000	0.000	0.000	0.000
11	11	6	1	8	1	30.000	2.650	500.000	0.000
12	12	6	2	0	1	0.000	0.000	0.000	0.000
13	13	7	1	6	1	25.000	1.910	300.000	0.000

Consulta ejecutada correctamente.

Ilustración 14: Tabla precarga

- **Tabla *ensayo***: guarda los datos generales de las pruebas efectuadas, por cada referencia de amortiguador. Tales como: la referencia de amortiguador, máquina usada, la fecha, etc.

```
select * from ensayo
```

100 %

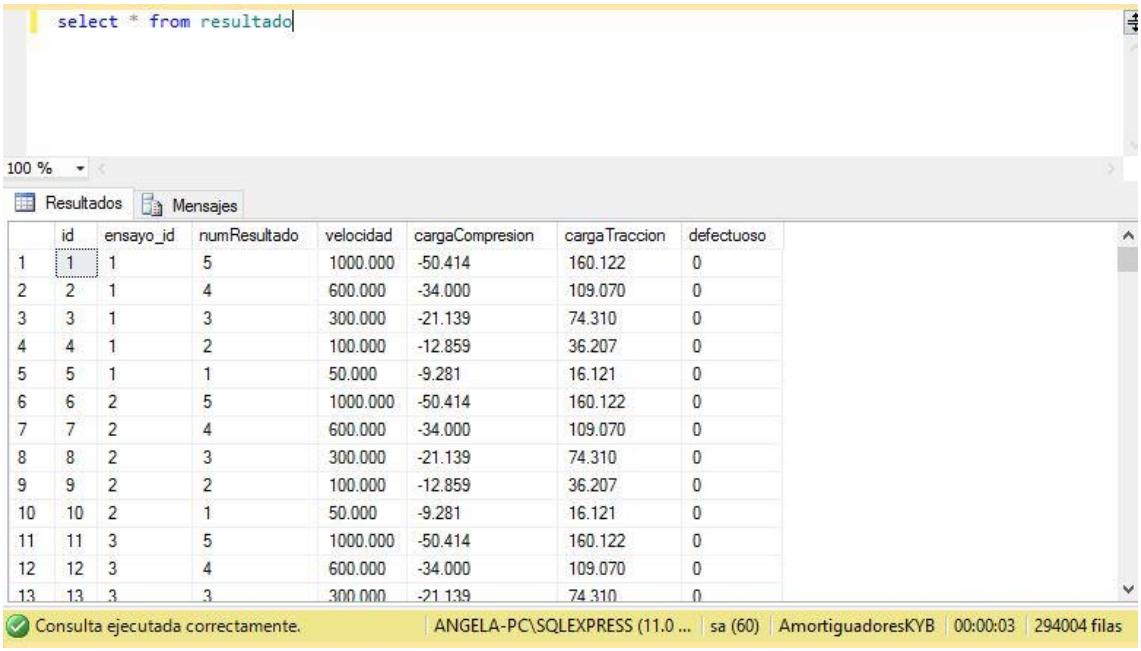
Resultados | Mensajes

	id	reglaje_id	maquina_id	referencia_id	numAmortiguador	gas	fecha	defectuoso
1	1	449	1	754	1	-24.559	2003-09-01 08:34:56.000	0
2	2	449	1	755	1	-24.559	2003-09-01 08:34:56.000	0
3	3	449	1	756	1	-24.559	2003-09-01 08:34:56.000	0
4	4	1176	2	1595	1	-6.000	2010-02-08 14:41:30.000	0
5	5	470	2	1509	1	1.000	2010-02-09 15:40:18.000	0
6	6	1157	2	1548	1	1.000	2010-02-09 15:50:20.000	0
7	7	1104	2	1588	1	-7.000	2010-02-15 15:51:44.000	0
8	8	1104	2	1588	2	-6.000	2010-02-15 15:53:15.000	0
9	9	1104	2	1588	3	-6.000	2010-02-15 15:54:29.000	0
10	10	369	2	1306	1	-4.000	2010-02-15 15:58:46.000	0
11	11	369	2	1306	2	-3.000	2010-02-15 15:59:59.000	0
12	12	369	2	1306	3	-4.000	2010-02-15 16:01:13.000	0
13	13	542	2	1520	1	-7.000	2010-02-15 16:33:08.000	0

Consulta ejecutada correctamente. | ANGELA-PC\SQLEXPRESS (11.0 ... | sa (60) | AmortiguadoresKYB | 00:00:01 | 90450 filas

Ilustración 15: Tabla ensayo

- **Tabla *resultado***: guarda en detalle los números obtenidos como resultado de las pruebas de calidad. Con cada amortiguador se realizan varios ensayos (condiciones del reglaje).



	id	ensayo_id	numResultado	velocidad	cargaCompresion	cargaTraccion	defectuoso
1	1	1	5	1000.000	-50.414	160.122	0
2	2	1	4	600.000	-34.000	109.070	0
3	3	1	3	300.000	-21.139	74.310	0
4	4	1	2	100.000	-12.859	36.207	0
5	5	1	1	50.000	-9.281	16.121	0
6	6	2	5	1000.000	-50.414	160.122	0
7	7	2	4	600.000	-34.000	109.070	0
8	8	2	3	300.000	-21.139	74.310	0
9	9	2	2	100.000	-12.859	36.207	0
10	10	2	1	50.000	-9.281	16.121	0
11	11	3	5	1000.000	-50.414	160.122	0
12	12	3	4	600.000	-34.000	109.070	0
13	13	3	3	300.000	-21.139	74.310	0

Ilustración 16: Tabla resultado

2.4 Creación de la base de datos nueva

A la hora de crear la base nueva, se debe concretar qué tipo de datos albergará cada campo.

Esto se elige así para esquivar futuras incoherencias.

Los tipos de datos que Microsoft SQL Server 2012 define son distintos a los de Microsoft Access [5]. La nueva base de datos dispone de los siguientes:

- **Enteros**: *int* y *bigint*.
  Se hace distinción entre enteros naturales (máximo valor absoluto: 2^{31}) y grandes enteros (máximo valor: 2^{63}). Los primeros se han concretado para atributos que usarán números enteros normalmente pequeños, como el número de ciclos de señal: "ciclos", en la tabla *condicion*.
  Los segundos se han concretado para atributos cuyo alcance se desconoce, como los identificadores: "id" de todas las tablas.
  Se acaban obteniendo más de 300.000 registros de datos en la tabla *resultado* tras la migración.

- **Booleanos, o lógicos**: *bit*.
  Se clasifican en el tipo "numérico exacto" dentro de SQL Server, pero son lo más similar a un valor verdadero/falso. En Microsoft Access son tipo sí/no. Pueden ser 0 o 1, y se han descrito para campos "serie" o "electronico" en la tabla *reglaje.* Indican si un amortiguador se fabrica o no en serie, y si es electrónico o no.

- **Decimales:** *decimal (10,3).*
  Como su nombre indica, es un tipo numérico con decimales. Se ha establecido una precisión de 10 cifras decimales, de las cuales sólo se mostrarán las 3 primeras. La razón es simple: actualmente se almacenan 2 cifras, que son suficientes para la medición de fuerza en estos ensayos.

- **Fecha y hora**: *datetime.*
  Puede almacenar una fecha y hora con formato: aaaa-MM-dd hh:mm:ss
  (año – mes – dia horas : minutos : segundos)

- **Cadena de caracteres**: *varchar(?).*
  Este tipo alberga cadenas de caracteres alfanuméricos, de longitud máxima la que se especifique entre paréntesis (?).
  Dependiendo de la finalidad del campo, se ha introducido una cantidad u otra. Por ejemplo, los nombres de referencias son cadenas cortas: se especifican 30 caracteres máximos.
  Mientras que los nombres de equivalencias pueden llegar a ser muy largos: se especifican 200.

  Debido a que el tipo *varchar* permite cualquier carácter, se procura que se emplee sólo si es absolutamente imprescindible; es decir, en atributos tales como los nombres.
  Se quiere evitar el error en la base original; en la cual los operarios podían introducir cualquier anotación textual en los datos.
  Hay alguna excepción como “numAmortiguador” en la tabla *ensayo.* Se permite puesto que en los datos anteriores ya había anotaciones y se quiere evitar su pérdida.

En la imagen se visualiza el apartado de diseño de la tabla *ensayo* en SQL Server Management Studio. Se aprecian los tipos de datos comentados.

Nombre de columna	Tipo de datos	Permitir val...
id	bigint	☐
reglaje_id	bigint	☐
maquina_id	bigint	☐
referencia_id	bigint	☐
numAmortiguador	varchar(30)	☐
gas	decimal(10, 3)	☐
fecha	datetime	☐
defectuoso	bit	☐
		☐

Ilustración 17: apartado de diseño de la tabla ensayo

SQL Server Management Studio es el software con interfaz gráfica incluido en el paquete de SLQ Server 2012 que permite gestionar las bases de datos cómodamente [6].

Gracias a él, se crea la nueva base de datos conforme al diagrama diseñado.

Se emplean las herramientas y opciones propias de la interfaz del programa, mientras que ciertas modificaciones de datos, consultas, desencadenadores (triggers)… se escriben en Lenguaje de Consulta Estructurado (SQL).

En las propiedades del identificador de todas las tablas, se impone que sea un entero grande único e incremental; comenzando por 1. Es decir, cada vez que se añada un registro a la tabla, su campo "id" será igual al último más uno. Y se le asigna la clave principal.

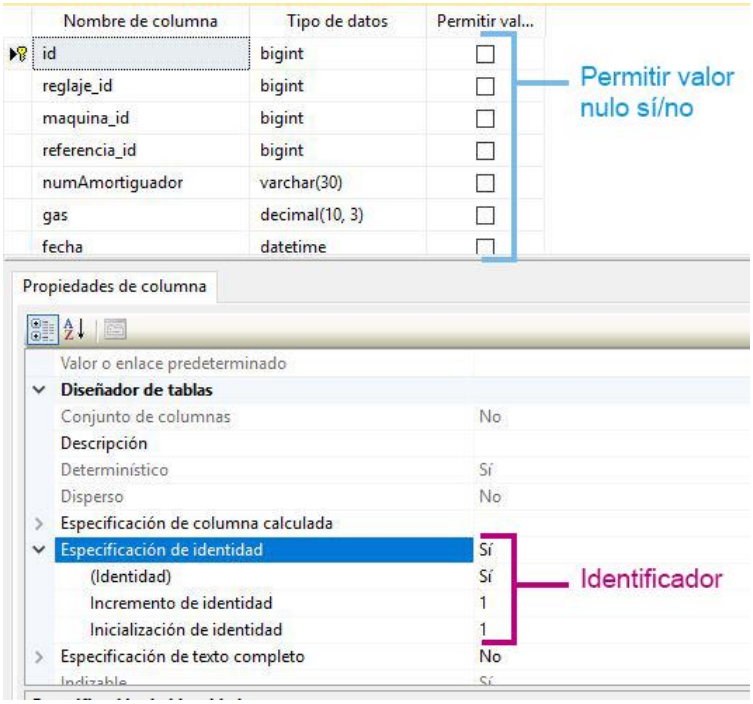


Ilustración 18: apartado de diseño en SQL Server Management Studio. Se marcan las opciones para campo identificador, y permiso de entrada de valores nulos

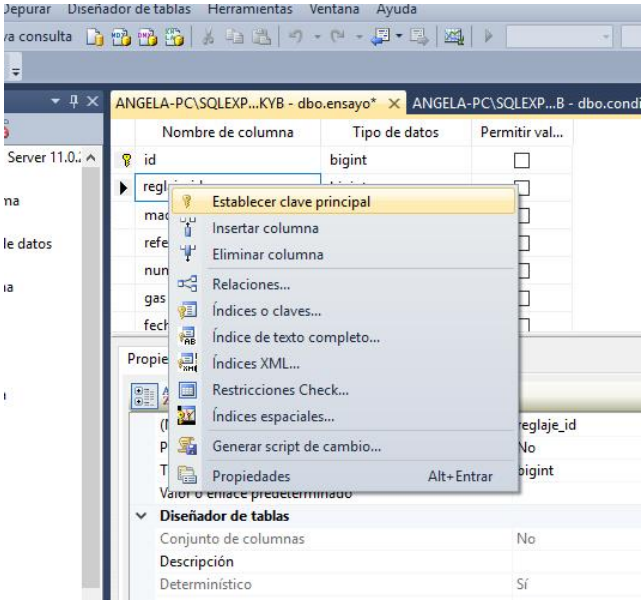


Ilustración 19: establecer una columna de tabla como clave principal

Los campos no aceptan valores NULL, como se ve en la Ilustración 18, que muestra los cuadros deseleccionados.

El atributo “reglaje_id” en *reglaje* sí acepta valores nulos, siendo una excepción. Se debe a que se empleará para hacer referencia al reglaje anterior, si se ha modificado; o a él mismo, si no. En el momento exacto de la creación del registro no se tiene el identificador (valor nulo), pero se modifica al instante mediante un desencadenador, o trigger.

Un trigger es una parte de código que se ejecuta automáticamente al suceder un evento en la base de datos.

Se escribe en SQL, de manera que, al insertar registros en esta tabla, el valor del campo “reglaje_id” sea igual al id del registro creado.

```
CREATE TRIGGER reglajeid ON reglaje AFTER INSERT AS
BEGIN
        SET NOCOUNT ON;
        DECLARE @regid INT;
        SELECT @regid = INSERTED.id FROM INSERTED;
        UPDATE reglaje SET reglaje_id = @regid WHERE id = @regid AND reglaje_id IS NULL;
END
```

Ilustración 20: código de trigger en la base de datos nueva

Se han hecho diversas modificaciones a lo largo del proyecto con el fin de ir solucionando inconvenientes que surgían, por lo que el primer diseño dista del definitivo.

En primeras versiones, campos como el nombre de las referencias o equivalencias, se restringieron para que fuesen valores únicos. Dado que luego se añadió el campo “borrado”, se

eliminó la restricción. Es posible que existan registros con mismo nombre, pero solamente uno tendrá valor 0 en el campo “borrado”.

El campo “comienzoCaptura” estaba pensado para la tabla ensayo en un inicio.

Se especificó así, puesto que en la visita a la fábrica, el software mostraba su cuadro de texto de antes de iniciar un ensayo. En una reunión posterior el cliente indicó su preferencia de tener dicho valor como un dato más de cada condición del reglaje. Finalmente está en la tabla *condicion*.

Actualmente, el ordenador de la fábrica que gestiona los ensayos posee una interfaz en la cual algunos valores se auto calculan. Estos son: frecuencia, amplitud o velocidad de la señal transmitida a la máquina durante el ensayo.

De modo que, al escribir un dato, los otros dos se completan gracias a una fórmula automáticamente.

En un primer momento, se pensó que el cálculo automático podía hacerse directamente en la base de datos.

Para ello se deja el tipo de la columna como ‘indefinido’ y su dato se calcula a partir de una fórmula. En la cual se involucran datos de otras columnas.

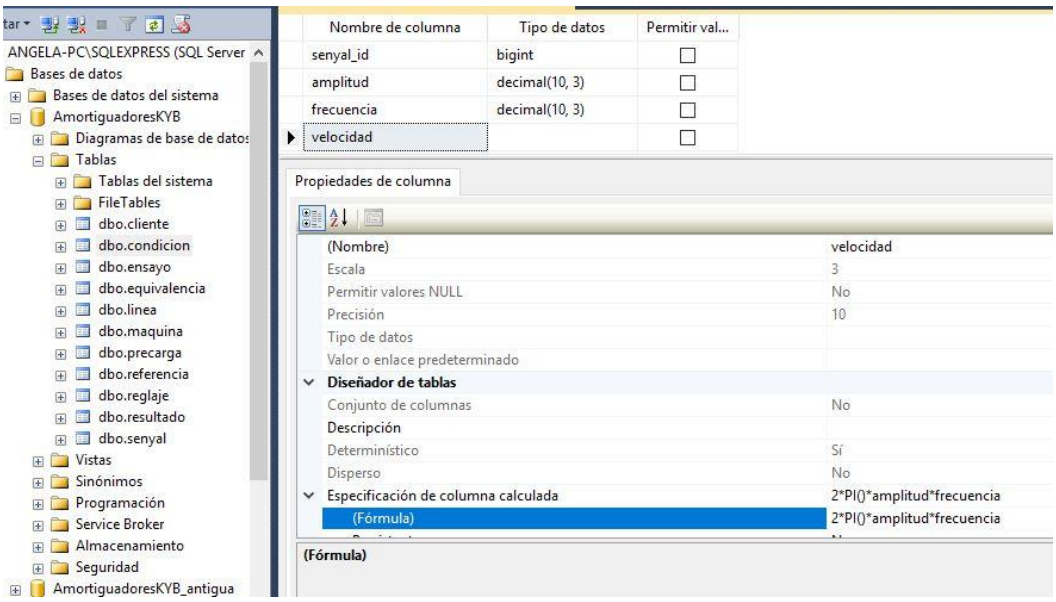


Ilustración 21: Especificación de columna "velocidad" calculada a partir de una fórmula, con "amplitud" y "frecuencia"

Surgieron algunos obstáculos:

- La fórmula era distinta si el tipo de señal era seno o rampa.
  Ello necesitaría programar una condición, mediante la cual se use una fórmula u otra dependiendo del valor en el campo “senyal” (seno o rampa).

Es un impedimento a largo plazo, ya que, si se incorporan nuevos tipos de señal, habría que cambiar también el código de la columna calculada.

- Fórmula utilizada para la señal seno:

$$v = 2 * \pi * a * f$$

v = velocidad en mm/s
a = amplitud en mm
f = frecuencia en Hz

- Fórmula para la señal rampa:

$$v = 4 * a * f$$

- Si se decide calcular otra columna: amplitud o frecuencia; aparecen divisiones. Y podrían insertarse ceros en el denominador. Esto se debe manejar programando en la propia fórmula, o se generan errores desde la base.

- Los usuarios desde la interfaz pueden seleccionar qué campo de los tres se calcula. Lo que supondría crear más código que cambie la estructura de la base (los tipos de datos de las columnas).
  Además, no es recomendable que un usuario tenga permisos para cambiar la estructura de una base desde la interfaz gráfica; ya que comprometería la integridad de los datos.

Finalmente se optó por resolver las cuestiones desde Java con la interfaz gráfica, programando condiciones sencillas ("if") y dejando las columnas en la base con un tipo fijo: datos decimales, de precisión 10 posiciones y 3 cifras.

La forma de establecer relaciones entre tablas es haciendo click derecho en la columna de una clave foránea → Relaciones → Agregar relación → En la edición de propiedades, seleccionar Especificación de tablas y columnas.

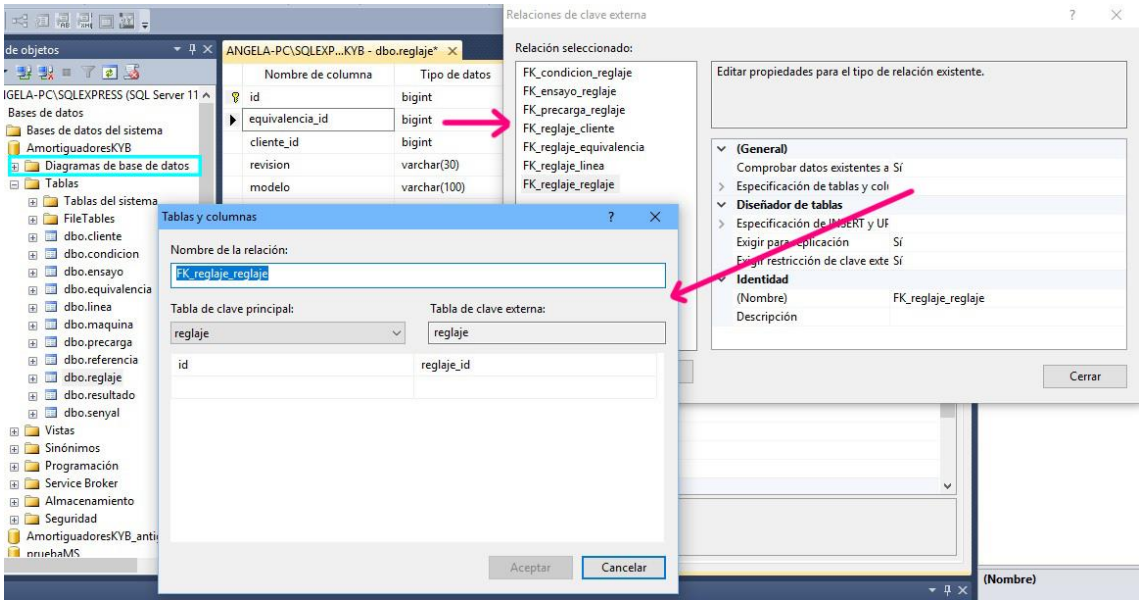


Ilustración 22: Especificar relaciones entre tablas mediante claves foráneas

En el apartado de “Diagramas de la base de datos” (marcado en azul, en el explorador de objetos de la ilustración anterior), se puede mostrar un diagrama que el software genera automáticamente tras aportar las relaciones entre tablas.

En la Ilustración 23 se presenta, y como es de esperar, es muy semejante al que ya se hizo manualmente en DIA.

SQL Server Management Studio simboliza las relaciones de identificación con dos iconos en los extremos: la llave y el infinito. Ello se debe a las relaciones entre tablas: 1 a N, generalmente. Significa que muchos registros (N) de un campo (simbolizado con el infinito), en una tabla pueden contener un valor que es clave primaria, un registro único (1), en otra tabla (simbolizado con la llave).

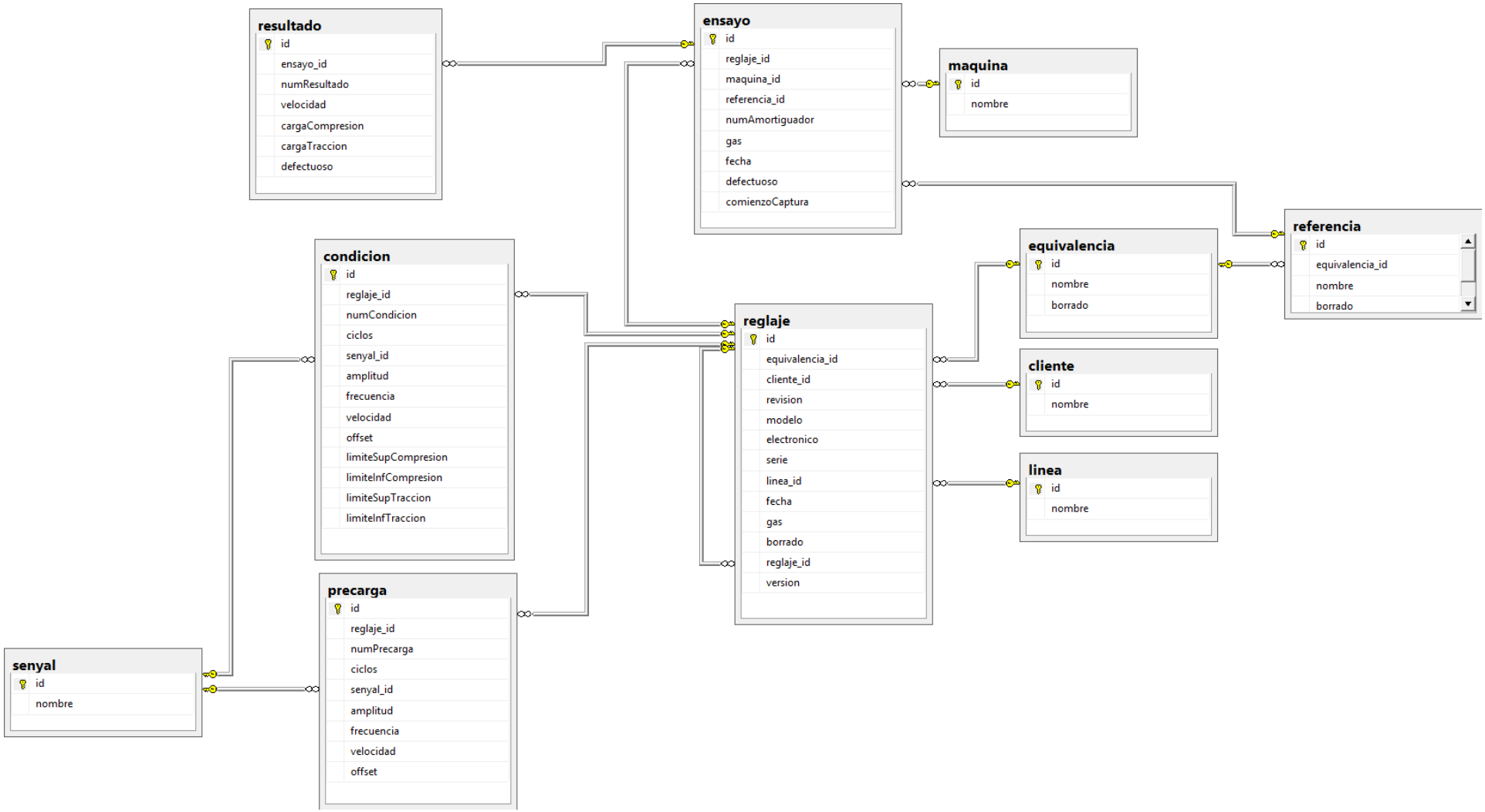


Ilustración 23: diagrama de relaciones de la base de datos nueva en SQL Server Management Studio

Finalmente, si se desea tener la estructura de esta base de datos nueva en la máquina del cliente (sin tener que repetir todos los pasos de creación allí); se puede generar un script desde este software que lo haga automáticamente.

El script contiene las líneas de código SQL necesarias para crear la estructura de esta base. Al generar el script, es muy importante marcar las opciones de creación de triggers, relaciones entre tablas, claves primarias, vistas, etc. Todo lo que la base requiera. Se pueden incluir los datos si se tienen.

Después, basta con ejecutarlo en la máquina del cliente.

2.5 Traspaso de formato de la base de datos original de MS Access a MS SQL Server

Para migrar los datos de la base original a la nueva, se lleva a cabo con las dos en formato SQL Server y de manera automática con programación Java. Para ello, primero se traspasa la base original de su formato Microsoft Access a formato Microsoft SQL Server.

Es posible hacerlo de varios modos.

Durante este proyecto se ha experimentado con dos; dado que se recibieron datos de la base en dos ocasiones.

Debido a las inconsistencias de datos en la base original, se solicitó asistencia de la empresa cliente. Esta envió una segunda base de datos; aunque resultó ser muy similar a la primera, con la diferencia de tener una cantidad de registros y resultados mucho mayor.

El análisis de las bases y la migración de información han sido más laboriosos y extendidos en tiempo de lo que se planeó en su inicio. Este tipo de improvistos se dan a menudo en empresas orientadas a proyectos.

Se aprovechó las dos ocasiones para hacer el traspaso de distinto modo y emplear el que mejor resultado generase.

En un primer momento, se exportó la base Access a formato Microsoft Excel; y de éste, a SQL Server. A pesar de parecer un poco enrevesado, fue simple puesto que ya se tenía el software instalado y se ejecutó con rapidez.

Sin embargo, se crearon tablas de errores de conversión con algunos registros incorrectos.

Con la segunda base, se descargó un software de Microsoft específico de migración de Access a SQL Server; que además habilitaba el traspaso de datos directamente desde la versión 97 obsoleta [7].

Este traspaso generó en todas las tablas un campo añadido de Timestamp, que puede ser utilizado para control en futuros cambios en la base de datos.

También una tabla, se intuye que temporal, que no coincidía con la estructura de ninguna otra (en campos).

De nombre ~TMPCLP360081 y 11 filas de información.

Al contener pocos datos, de los cuales algunos eran pruebas, se ignoró.

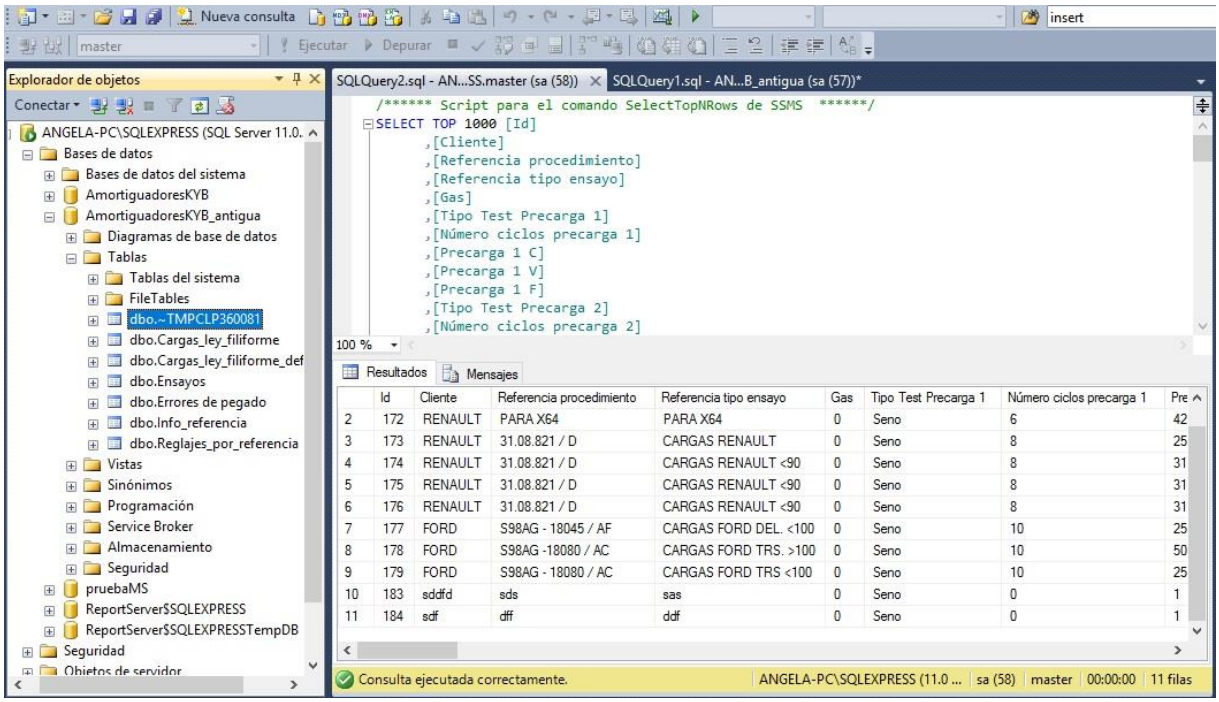


Ilustración 24: tabla temporal y datos

2.6 Migración de datos de la base de datos original a la base de datos nueva

2.6.1 Clases en Java para la migración

Se procedió a la programación de una clase en Java que moviese los datos de una base SQL Server, a otra.

Dada la complejidad y la cantidad de acciones a realizar, causadas por la incoherencia de los datos; se creó un paquete con diferentes clases.

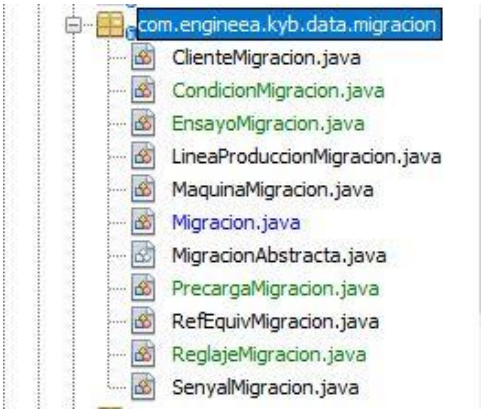


Ilustración 25: Paquete con las clases de migración

Las clases destacadas son:

- ✓ ***Conexion**:* una clase que abre una conexión con la base de datos, gracias a la API de Microsoft, JDBC para SQL Server [8]. Para ello, lógicamente, el servidor de SQL Server debe estar activo y a la escucha. Esto se hace gracias al administrador de SQL Server incluido en su propio paquete de instalación.
  Éste sirve además para asegurar que los puertos establecen la conexión correctamente y pueden transportar información; ver en las propiedades el número de puerto exacto, modificar sus permisos, etc.

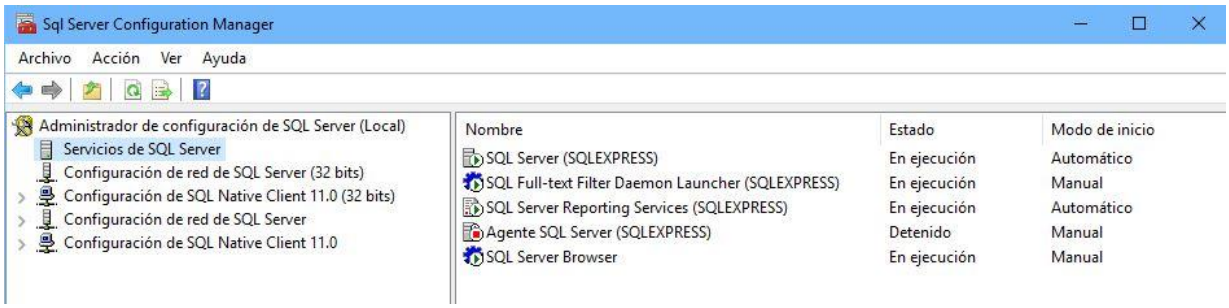


Ilustración 26: Administrador del servidor MS SQL Server

Debido a que se realizarán conexiones a las dos bases de datos en todo el proyecto, esta clase se guarda en un paquete especial para ello llamado: com.engineea.kyb.data.

- ✓ ***Migracion***: la clase que contiene el método principal a ejecutar (*main*). Irá creando objetos del resto de clases y ejecutando un método de cada una de ellas, llamado *migrar().*

- ✓ ***MigracionAbstracta***: una clase abstracta que implementa métodos que se usarán repetidamente; usando la conexión mencionada en el primer guión.

- ✓ ***ClienteMigracion***: es una clase cuyo método *migrar()* toma los nombres de clientes de la tabla de la base original *Info_por_referencia* y los inserta en la tabla *cliente* de la base nueva.

- ✓ ***SenyalMigracion***: introduce los tipos de señal en la tabla *senyal* de la base de datos nueva.

- ✓ ***MaquinaMigracion***: se conecta con la base existente para adquirir los tipos de máquina que aparecen en los registros de resultados. Después con *migrar()* los inserta en la tabla *maquina* de la base de datos nueva.

- ✓ ***LineaProduccionMigracion***: inserta las líneas de producción en la tabla *línea* de la base de datos nueva.

- ✓ ***RefEquivMigracion***: toma las equivalencias de la base original y las separa en referencias únicas, creando los registros necesarios en sus correspondientes tablas *equivalencia* y *referencia*.

- ✓ ***ReglajeMigracion***: crea los registros de reglajes por equivalencias en la tabla *reglaje*.

- ✓ ***PrecargaMigracion***: añade los registros de precarga de cada reglaje a la tabla *precarga*.

- ✓ ***CondicionMigracion***: añade los registros de condiciones de cada reglaje a la tabla *reglaje*.

- ✓ ***EnsayoMigracion***: completa las tablas *ensayo* y *resultado.*
  Inserta en la tabla *ensayo* los datos generales de los tests, como son la referencia con que se ha hecho, el reglaje aplicado, la fecha, etc.
  Inserta en la tabla *resultado* los resultados concretos de fuerza en cada ensayo, que se distinguen por velocidades.

2.6.2 Programación e incidencias

Se detallan a continuación la programación e inconveniente que han ido surgiendo en algunas de las clases.

1. MigracionAbstracta

En algunas clases se repite la inserción de un único valor de nombre en una tabla. Por ello se creó esta clase y se le añadió el método: *insertarString(String nombre, String nombre de la tabla*). Los parámetros de entrada son Strings (cadenas de texto) del nombre que se va a insertar en la fila de la tabla, y el nombre de la propia tabla. El dato de salida del método es un número tipo Long con el índice (id) del registro recién insertado. El resto de clases implementará esta interfaz para usar el método.

MigracionAbstracta hace una consulta de inserción (Insert en lenguaje SQL) en la base de datos nueva. En todo el proyecto y casi sin excepción, se utilizarán los "prepared statements" [9] para mayor seguridad.

Los Prepared Statements son declaraciones que evitan la concatenación de cadenas de texto indeseadas en la consulta. Por ejemplo, el borrado de las tablas (Drop en SQL).

Serán muy útiles en la interfaz gráfica, donde el operario puede introducir cualquier texto.

```
public static Long crearCliente(String nombre) {
    Long id = null;
    String sql = "INSERT INTO cliente (nombre) VALUES (?)";
    try {
        PreparedStatement statement = Conexion.Conexion.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
        statement.setString(1, nombre);
        int filas = statement.executeUpdate();
        if (filas == 1) {
            try {
                ResultSet generatedKeys = statement.getGeneratedKeys();
                generatedKeys.next();
                id = generatedKeys.getLong(1);
            } catch (SQLException e) {
            }
        }
    } catch(SQLException e) {
    }
    return id;
}
```

Ilustración 27: Ejemplo de código en el que se insertan valores en una tabla.
Se usa Prepared statements para evitar concatenar consultas indeseadas.

2. *ClienteMigracion*

Para conseguir todos los nombres de clientes se hace una conexión a la base anterior ya en formato SQL Server, y se realiza una consulta escogiendo los valores distintos (Select Distinct en SQL) del campo *Cliente* en la tabla *Info_referencia*.

Después, con *insertarString()*, se introducen en la tabla *cliente.*

Al insertar un registro en una tabla, se genera automáticamente su id (como se había especificado en el diseño y creación de la base). Se necesita dicho id para insertarlo después en campos de otras tablas. Por ejemplo: en la tabla *reglaje* hay un campo de "cliente_id".

Para conseguir el id, se añade en la declaración de consulta (el "statement") un parámetro: RETURN_GENERATED_KEYS. Que devuelve un número con el identificador recién generado.

En una tabla hash de Java, se van añadiendo el nombre del cliente asociado a su identificador.

Las tablas hash son tablas asociativas, que mapean unos índices (keys) con unos valores. En este caso, los índices son cadenas de texto (Strings) y los valores, son los identificadores numéricos (tipo Long).

Clave (key)	Valor
“AUDI”	2
“GENERAL MOTORS	10
“TOYOTA”	3
…	

Tabla 2: ejemplo de tabla hash

De esta forma, es sencillo en otras clases poder consultar qué identificador pertenece a un nombre de cliente.

```
public class ClienteMigracion extends MigracionAbstracta {

    private final Map<String, Long> tablaCliente = new HashMap<>();

    @Override
    public Map<String, Long> migrar() throws SQLException {
        Statement statement = Conexion.ANTIGUA.createStatement();
        String sql = "SELECT DISTINCT Cliente FROM Info_referencia ";
        ResultSet setClientes = statement.executeQuery(sql);
        while (setClientes.next()){
            String nombreCliente = setClientes.getString("Cliente");
            Long idCliente = insertarString("cliente","nombre",nombreCliente);
            tablaCliente.put(nombreCliente, idCliente);
        }
        return tablaCliente;
    }
```

Ilustración 28: ejemplo de método migrar() que devuelve la tabla hash de clientes

La salida al método *migrar()* de *ClienteMigracion* es la tabla hash de Java con los nombres de clientes y sus identificadores en la base nueva.

3. SenyalMigracion

El cliente afirmó que normalmente emplean sólo señales de tipo seno en sus ensayos, y excepcionalmente, señales tipo rampa.

Puesto que la base de datos original tiene variantes en nomenclatura de las mismas señales; se decidió no hacer una conexión con la base original. Creando estas dos señales de modo manual e insertándolas en la base nueva.

Los valores que se encontraban en la base original eran del estilo: *seno, Seno, _ seno,_ Seno, rampa, Rampa, triángulo*, etc.

El método *migrar()* devuelve una tabla hash con los tipos de señales y sus identificadores.

4. *MaquinaMigracion*

Selecciona los valores distintos del campo *Maquina* de la tabla *Cargas_ley_filiforme;* y los inserta en la tabla *maquina* de la base nueva.

Con los datos del cliente hay dos máquinas: *MTS10255* y *MTS10278.*

En los resultados, se incluyen tanto los amortiguadores defectuosos como los correctos en la misma tabla; y dejar un atributo para definir el estado.

Sin embargo, la tabla que contiene resultados defectuosos *Cargas_ley_filiforme_defectuoso* no alberga un campo que identifique a la máquina. Por lo que se añade a la tabla *maquina* de la base de datos nueva otro valor más a mano, con el nombre "Otra". En el que encasillar estos resultados de amortiguadores defectuosos.

El método devuelve una tabla hash con los nombres de máquinas y sus identificadores.

5. *LineaProduccionMigracion*

Tiene el mismo inconveniente que los tipos de señal: las líneas en la tabla *Info_referencia* de la base de datos original no estaban escritas siguiendo un formato.

Sabiendo que existen 14 líneas de producción en la fábrica, se crean unas cadenas de texto con formato *Línea ?*, donde ? es un número entero del 1 al 14. Se añade una línea más llamada *Otros* para aquellos amortiguadores fabricados en varias líneas.

El método *migrar()* inserta estos nombres en la tabla *linea* de la base de datos nueva y devuelve una tabla hash con los nombres e identificadores.

6. *RefEquivMigracion*

Esta clase ha sido una de las que más incidencias ha supuesto en la migración a causa de los datos hallados en la base original.

La forma de proceder del método *migrar()* consiste en seleccionar el campo "Referencia AP" y "Referencia AP amortiguador" de una unión (JOIN en lenguaje SQL) de las tablas *Info_referencia* y *Reglajes_por_referencia*.

Lo esperado es que ambas contengan las mismas equivalencias.

Sin embargo, no fue así: unos pocos valores no estaban en ambas tablas a la vez (aparecían en una tabla y no en la otra), y había referencias que formaban parte de diferentes grupos al mismo tiempo (en la misma tabla o en las dos).

Por ejemplo, la referencia “6856” aparece a la vez en dos grupos: 6514/6515/**6856**/7770… y en 6555/**6856** en la tabla *Info_referencia*.

La referencia **5332** aparece suelta en la tabla *Info_referencia* y agrupada en la equivalencia **5332**/1999/1998/1997 en la tabla *Reglajes_por_referencia.*

Se pensó en mezclar las equivalencias, pero no fue posible dado que tenían distintas características en los reglajes.

La solución es que, antes de insertar una referencia en la tabla *referencia*, se comprueba si ya se ha introducido antes. Si la respuesta es afirmativa, se actualizará más adelante el campo “equivalencia_id” en la tabla *referencia*; para que apunte a uno de los grupos.

Algunas referencias tienen notación: referencia_LEY ?.

Donde ? es un número entero. Por ejemplo: 7162_LEY1, 7162_LEY2.

Algunas equivalencias albergaban mismas referencias en un grupo, con distinta “ley”: 7161/**7162_LEY1**, 7161/**7162_LEY2**.

Tras reuniones con el cliente, éste expresó que ya no operaban con el tema de leyes, por lo que no era un hecho fundamental; pero que se procurase perder el menor número de registros posible. Por ello se decide tratar estas equivalencias como referencias únicas, cambiando la barra “/” por un guión bajo “_”.

Exceptuando estos eventos, el funcionamiento normal es:

1) Crear un registro en la tabla *equivalencia* con el nombre completo. Si contiene la palabra ley, se refactoriza (edita) el nombre.

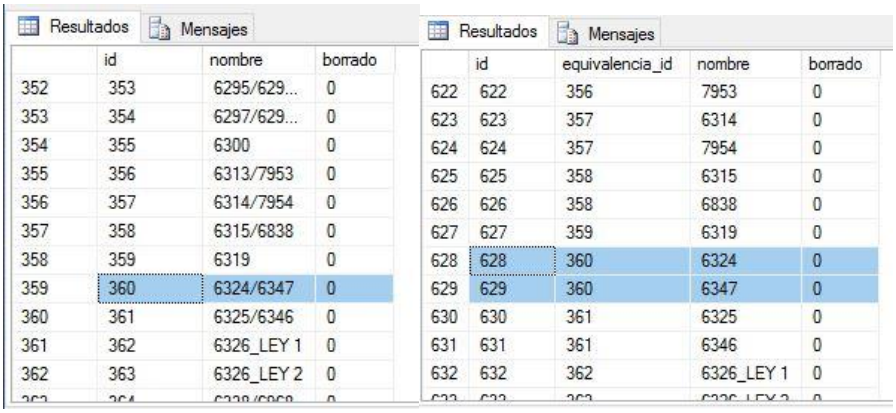
2) Separar el nombre de la equivalencia por barras, en referencias.

3) Intentar crear sendos registros en la tabla *referencia.* Antes de insertarlos, se comprueba si ya existen. En cuyo caso, se actualiza el atributo “equivalencia_id”, con el identificador obtenido en el paso 1.

4) Si no estaba en la tabla *referencia* (es el funcionamiento normal), se inserta directamente con el dato del nombre de la referencia y el id de la equivalencia.

En la Ilustración 29: Tablas equivalencia (izquierda) y referencia (derecha) se resalta el ejemplo de la equivalencia 6324/6347.

Consiste en un grupo de dos referencias: la 6324 y la 6347. El campo “equivalencia_id” de las dos, en la tabla *referencia* apunta al identificador de la equivalencia (360) en la tabla *equivalencia*.



Resultados | Mensajes

	id	nombre	borrado
352	353	6295/629...	0
353	354	6297/629...	0
354	355	6300	0
355	356	6313/7953	0
356	357	6314/7954	0
357	358	6315/6838	0
358	359	6319	0
359	360	6324/6347	0
360	361	6325/6346	0
361	362	6326_LEY 1	0
362	363	6326_LEY 2	0

Resultados | Mensajes

	id	equivalencia_id	nombre	borrado
622	622	356	7953	0
623	623	357	6314	0
624	624	357	7954	0
625	625	358	6315	0
626	626	358	6838	0
627	627	359	6319	0
628	628	360	6324	0
629	629	360	6347	0
630	630	361	6325	0
631	631	361	6346	0
632	632	362	6326_LEY 1	0

Ilustración 29: Tablas equivalencia (izquierda) y referencia (derecha)

En posteriores reuniones con el cliente, éste comento que se podían ignorar las referencias con una “P” al final de su nombre, puesto que eran pruebas de los operarios. Y se siguió su recomendación.

7. ReglajeMigracion

La tabla *reglaje* albergará datos generales de las equivalencias, por lo que toma todos los datos de la tabla *Info_referencia* de la base original, y el dato de gas (booleano; verdadero/falso), de la tabla *Reglajes_por_referencia*.

Como se comenta en el apartado anterior, no coinciden algunas equivalencias, pero se solventa añadiendo un método redundante, que busca los datos con el nombre de cada referencia única. Se hace así por la referencia que aparece suelta en la tabla *Info_referencia* y agrupada en *Reglajes_por_referencia*. Con otras referencias no ha sido posible conseguir los datos; por lo que existen en las tablas de la base nueva *referencia* y *equivalencia*, pero no tienen reglaje asociado en la tabla *reglaje*.

Dado que todos los registros son de nuevo ingreso en la base, el dato en “reglaje_id” será igual que el dato en “id”. Este campo está pensado para que cuando se modifique un reglaje, se cree un nuevo registro, con su identificador nuevo, pero con un “reglaje_id” igual al modificado.

Gracias a que se han ido guardando las tablas hash, es sencillo insertar en *reglaje* los identificadores de los correspondientes “cliente”,”senyal”, ”línea”, etc. Buscando por los nombres. Véase un ejemplo de los datos en la Ilustración 12: Tabla reglaje.

El método *migrar()* devuelve una tabla hash que relaciona los nombres de las equivalencias con el identificador de su reglaje.

8. PrecargaMigracion

Para hacer la migración hacia la tabla *precarga* se necesita haber creado antes el reglaje que corresponde, puesto que uno de sus campos (“reglaje_id”) establece la relación entre las dos tablas.

Existen dos precargas, como máximo, en la base de datos original. Para lo cual existen atributos como: “Tipo test precarga 1”, “Test 1 V”, “Test 1 F” … Que se referirían a las características de la precarga número 1. En este ejemplo serían el tipo de señal, la velocidad y la frecuencia respectivamente.

La precarga número 2 tiene los mismos atributos: “Tipo test precarga 2”, ”Test 2 V”, “Test 2 F”…

El funcionamiento es el siguiente:

1) Se hace un bucle que recorre la tabla hash de reglaje y selecciona los datos de cada equivalencia, campo “Referencia AP amortiguador”,de la tabla *Reglajes_por_referencia.*

2) Dentro de éste, otro bucle repite una operación con los enteros 1 y 2. Concatenando en la consulta este número en el título de dichos campos.

3) Si se encuentran datos y no son nulos (NULL), se añaden a una lista (ArrayList). De modo que antes de insertar en la nueva tabla *precarga* se comprueba que el tamaño de la lista es el esperado: de 8 elementos. Que son: id del reglaje, número de precarga, número de ciclos de señal, tipo de señal, amplitud de señal, velocidad, frecuencia y offset.

9. CondicionMigracion

La migración de las condiciones es homóloga a la de precargas; con la diferencia de que se trata de números del 1 al 14, y existen cuatro campos de datos más: los límites superior e inferior de fuerza en compresión y tracción.

Consecuentemente, la lista tendrá un mayor tamaño.

En el apartado 2.2: Análisis de la base de datos original, ya se menciona que muchos campos en *Reglajes_por_referencia* tienen ceros. De lo cual se interpreta, que son condiciones sin uso.

Esto se hace aún más evidente, si se entiende el significado de campos como “frecuencia”, ”velocidad”, ”amplitud” de la prueba; que deberían ser distintos de 0 generalmente.

El problema ha estado en decidir qué registros se ignoraban.

En algunos reglajes, los datos de condición eran ceros, pero los de precarga no. O viceversa.

Ocurría también, que algunos tenían números correctos en los campos de límites, pero no en sus velocidades, frecuencias o amplitudes. Y viceversa. Se han dado muchas casuísticas; tanto en precargas como condiciones. Por lo que, se decidió migrar todos los registros.

Sin embargo, de ahora en adelante en la nueva base, se ignorarán las condiciones introducidas desde la interfaz cuyos datos sean todos igual a 0.

Más adelante en el proyecto, se vio que el orden numérico de las condiciones en el reglaje no coincidía con el orden obtenido de los ensayos. En algunos es ascendente, en otros descendiente, y en ocasiones no tienen orden.

Es decir, no correspondía el número de condición y la velocidad en el reglaje, con el número de ensayo y la velocidad aplicada (para la referencia que usa ese reglaje).

Puesto que la máquina de ensayos procede de menor a mayor velocidad; se decide ordenar las condiciones de ese modo, antes de migrar los datos.

Se hace mediante un simple algoritmo que recorre un array (estructura) de números, almacena cada uno y compara con el siguiente.

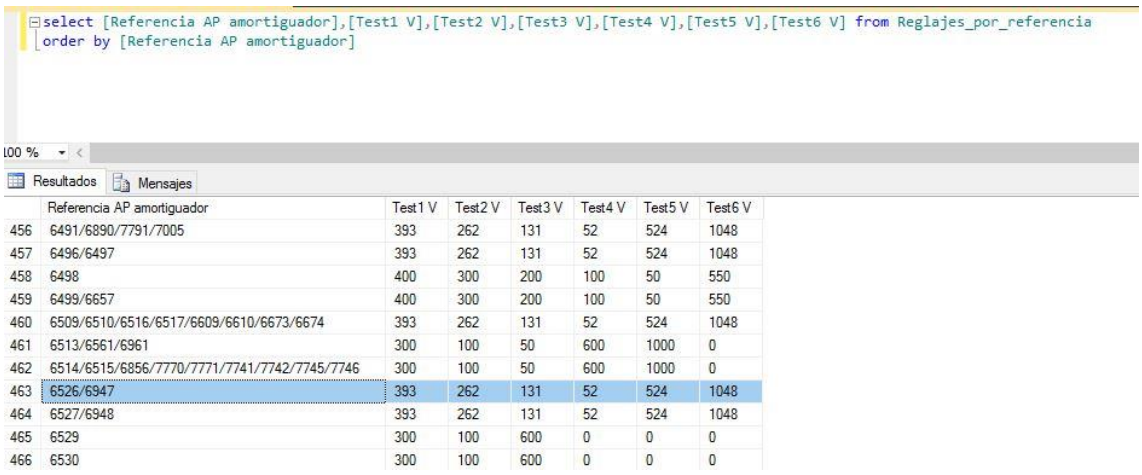
```
select [Referencia AP],V1,V2,V3,V4,V5,V6 from Cargas_ley_filiforme
```

100 %

Resultados | Mensajes

	Referencia AP	V1	V2	V3	V4	V5	V6
1	6514/6515/6856	1000	600	300	100	50	0
2	8357	600	300	50	0	0	0
3	8108	550	300	50	0	0	0
4	8222/8223_LEY 1	600	300	100	0	0	0
5	8340	600	300	50	0	0	0

Ilustración 30: velocidades de algunas condiciones en Cargas_ley_filiforme



```
select [Referencia AP amortiguador],[Test1 V],[Test2 V],[Test3 V],[Test4 V],[Test5 V],[Test6 V] from Reglajes_por_referencia
order by [Referencia AP amortiguador]
```

100 %

Resultados | Mensajes

	Referencia AP amortiguador	Test1 V	Test2 V	Test3 V	Test4 V	Test5 V	Test6 V
456	6491/6890/7791/7005	393	262	131	52	524	1048
457	6496/6497	393	262	131	52	524	1048
458	6498	400	300	200	100	50	550
459	6499/6657	400	300	200	100	50	550
460	6509/6510/6516/6517/6609/6610/6673/6674	393	262	131	52	524	1048
461	6513/6561/6961	300	100	50	600	1000	0
462	6514/6515/6856/7770/7771/7741/7742/7745/7746	300	100	50	600	1000	0
463	6526/6947	393	262	131	52	524	1048
464	6527/6948	393	262	131	52	524	1048
465	6529	300	100	600	0	0	0
466	6530	300	100	600	0	0	0

Ilustración 31: velocidades de algunas condiciones en Reglajes_por_referencia

10. EnsayoMigracion

La clase toma los registros de las tablas *Cargas_ley_filiforme* y *Cargas_ley_filiforme_defectuoso.*

Puesto que la segunda tabla no tiene el campo máquina, a los registros que provengan de ella se les atribuye el identificador de máquina asociado a “Otra”.

La tabla *ensayo* tiene un campo denominado “defectuoso”, de forma que los registros procedentes de *Cargas_ley_filiforme* tengan este campo serán igual a 0 (no son defectuosos), y los de *Cargas_ley_filiforme_defectuoso* obtendrán un 1 (sí son defectuosos).

Una vez creados los registros de la tabla *ensayo*, se insertar en la tabla *resultado* los distintos resultados, del mismo modo que se hacía con las condiciones y el reglaje.

En cuanto a las velocidades, se ha cambiado el número de resultado (campo “numResultado”), para que coincida con el orden de condiciones del reglaje.

Cada resultado guardado pertenece a una equivalencia. Pero una de las peticiones del cliente es poder filtrar los resultados por referencia.

Lo que se ha hecho es dividir los registros por referencia, repitiendo los datos del resto de campos.

Esto por un lado cumple la petición del cliente, pero por otro genera gran carga de datos y, por tanto, la migración tarda varios minutos en completarse.

También hay que tener en cuenta que la base almacena unos 30.000 registros con ensayos hechos desde el año 2003 (atendiendo al campo “fecha”).

En la segunda versión de la base proporcionada por el cliente, se encontraron en la tabla *Cargas_ley_filiforme* cuatro registros con el campo “Referencia AP” nulo. Lo que impide que se determine con certeza de qué amortiguador se tratan los datos de los registros. Se guardaron aparte, pero se ignoran en la migración.

3. Interfaz Gráfica de Usuario y modo de ejecución

3.1 Objetivos

El objetivo fundamental es lograr un software que actúe de traductor entre las instrucciones del operario de la fábrica (usuario), la base de datos y la máquina de ensayos.

De ahora en adelante, se referirá a la base de datos, como la base de datos nueva creada para este proyecto y cargada con los datos de la base original.

En detalle, la interfaz debe ser capaz de:

- Consultar resultados de ensayos anteriores.

- Enviar las órdenes a un dispositivo controlador, que sirve como puente entre el ordenador y la máquina.

- Gestionar las referencias contenidas en la base de datos: consultar, eliminar y añadir referencias.

- Gestionar las equivalencias contenidas en la base de datos: modificar y crear equivalencias.

- Gestionar los reglajes contenidos en la base de datos: modificar, eliminar y crear reglajes.

Se toman estas operaciones básicas de fotografías obtenidas en el software de la fábrica:

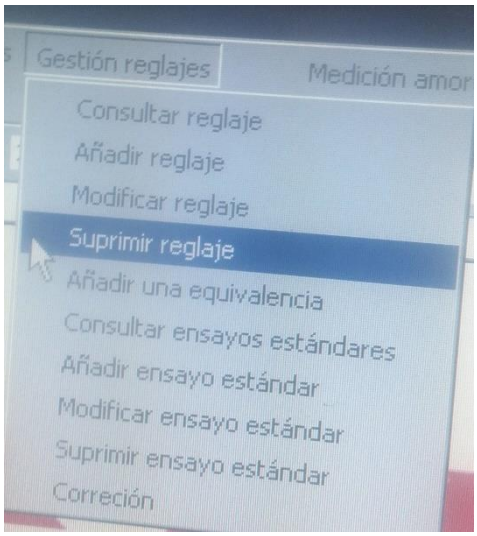


Ilustración 32: fotografía del software actual de la fábrica con operaciones básicas de reglajes

3.2 Requisitos de diseño la interfaz

Algunos requisitos generales del programa son:

1. **Buena visibilidad.**

Gracias a la visita a la fábrica, se sabe que los ordenadores están protegidos por unas pantallas que dificultan la visión; por lo que se decide emplear fuentes de letra y elementos grandes siempre que sea posible en la interfaz. También, por el hecho de no conocer con exactitud la resolución.

Esto además sería una ventaja a largo plazo, ya que facilitaría el uso si se instalan pantallas táctiles en un futuro (algo muy común en la industria).

2. La interfaz se desarrollará bajo un único marco.

En un primer momento se piensa en crear la interfaz mediante distintos JFrames (marcos), de modo que cada pantalla fuese una ventana; y sólo estuviese activa una cada vez.

Se desechó la idea por los problemas que surgían; como el traspaso de datos entre pantallas.

Otra idea planteada era usar layouts. Un layout es la manera que tiene Java de distribuir los elementos en una interfaz gráfica.

Existen diversos layouts, se pensó que el layout card podría ser útil. Este layout usa un sólo frame (marco) y se puede seleccionar qué panel se muestra en cada momento (ocultando el resto como en una baraja de cartas).

Pero no era muy cómodo puesto que cada pantalla de la interfaz necesita un tamaño distinto.

Pantallas como la de detalles del reglaje son extensas (Ilustración 47: interfaz Nuevo reglaje – Detalles), y otras son reducidas, como la selección de una referencia en un buscador (Ilustración 54: Modificar reglaje - Seleccionar referencia).

La solución es crear un único frame principal, del cual se originarán unas ventanas de diálogo (JDialogs) modales, cada una con sus características propias.

Modal quiere decir, que una vez que se activa un JDialog no se puede pinchar con el ratón fuera de él (en el programa activo).

Cada pantalla supone un JDialog, y sólo hay una en el frente, contando con botones de "cancelar" o "siguiente/aceptar" en todos los diálogos. Detrás queda situada la ventana del menú principal (la primera en aparecer al ejecutar el programa).

El cambio de unas ventanas a otras generalmente no se hace cerrándolas, sino ocultándolas. De este modo, cuando se cancela una acción, no se pierden los datos de la ventana anterior.

Se hace cambiando la propiedad de visibilidad con el método: *setVisible(boolean)* de los JDialog. Donde *boolean* puede ser *true* (verdadero = visible), o *false* (falso = oculto).

Todas las ventanas son de tamaño fijo, marcado por el método: *setResizable(false).*

La librería swing ofrece muchos métodos de fijación de características de los elementos gráficos.

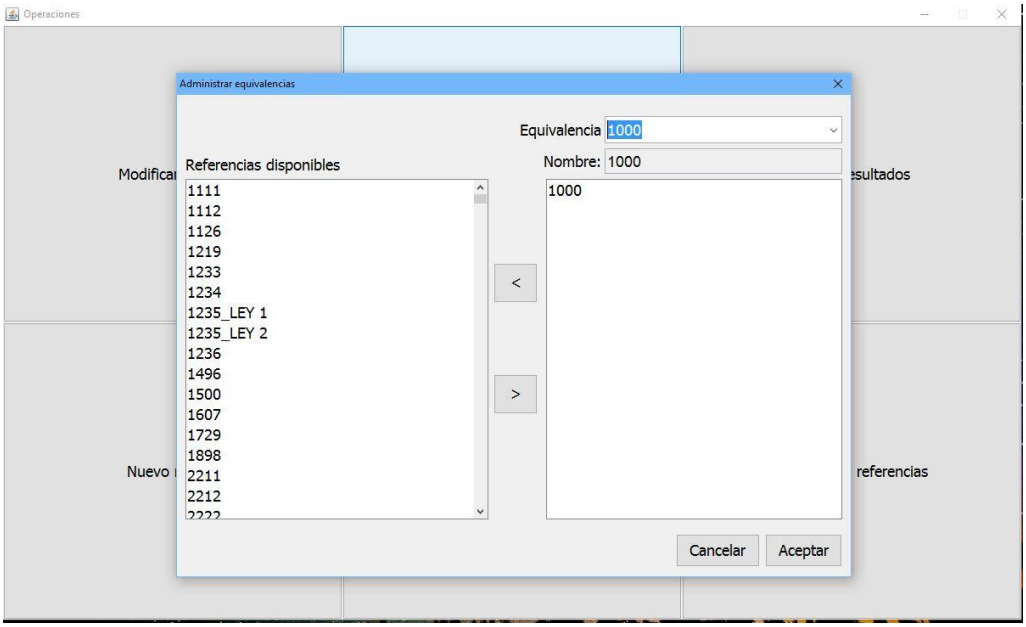


Ilustración 33: Ejemplo de la ventana principal con un diálogo modal delante

3. Existirá un control de errores en la inserción de datos

Se utilizan pequeñas ventanas emergentes (pop-ups) para controlar la entrada de datos incorrectos antes de que estos entren en conflicto con la base de datos.

En Java estas ventanas son los JOptionPane.

Los JOptionPane pueden ser de muchos tipos, pero en este proyecto se han utilizado dos:

- ✓ Los **mensajes informativos**, con un solo botón de aceptar. Generalmente se utilizan para informar de errores.
  Como ejemplo, en la siguiente ilustración, se ha escrito el decimal 2.5 en el campo "Número de ciclos" de la interfaz. Se quiere que el campo guarde valores enteros, por lo que se informa al usuario en un mensaje de error con un elemento JOptionPane de Java.

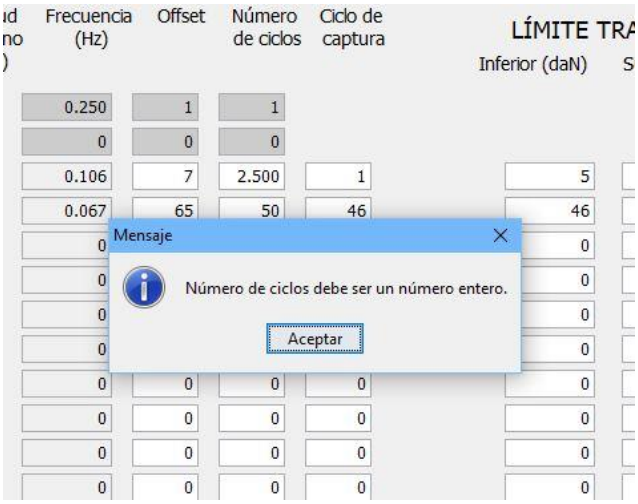


Ilustración 34: ejemplo de mensaje informativo.

- ✓ Los **mensajes de interrogación**, que formulan una pregunta al usuario con dos posibles respuestas de tipo: sí/no.
  Como ejemplo, en la siguiente ilustración se ha escrito un nombre de cliente que no aparece en la lista desplegable, por lo que un mensaje pregunta al usuario si se desea crear uno nuevo. Cada botón tendrá dará lugar a su acción correspondiente.

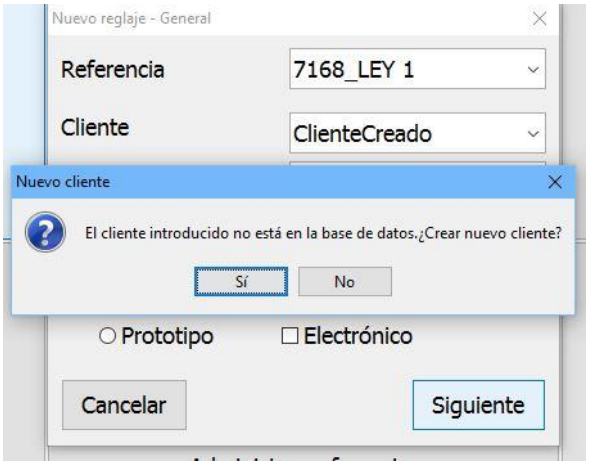


Ilustración 35: ejemplo de mensaje de interrogación

4. **Se utilizará una estructura ordenada de paquetes para el proyecto y el modelo vista - controlador**

El proyecto de la interfaz se crea en Gradle, para poder descargar las librerías necesarias al ejecutarlo; sin necesidad de tenerlas almacenadas previamente en la máquina local.

Está dividido en una serie de paquetes. Su nomenclatura sigue un formato:

dominio . empresa desarrolladora . empresa cliente . función

Ejemplo: com.engineea.kyb.util

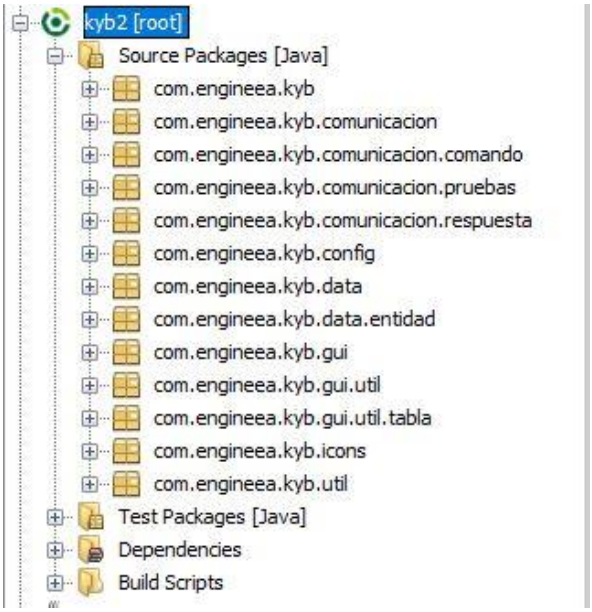


Ilustración 36: paquetes de clases en el proyecto

Entre estos paquetes destacan:

- ✓ ***kyb***: contiene el método *main*. Es el que corre la aplicación.

- ✓ ***comunicacion***: son todos los paquetes relativos a la simulación de conexión con el dispositivo de adquisición de datos; para lanzar ensayos y obtener resultados.

- ✓ ***config***: contiene lo necesario para configurar con éxito la conexión a la base de datos.

- ✓ ***data y data.entidad***: en *data* están las clases que efectuarán la conexión a la base y *data.entidad* es un conjunto de clases que representan a cada una de las tablas.
  Cada clase de *entidad* tiene métodos para crear, modificar, borrar, etc. objetos o listas de objetos que representan a los datos de la base. Es semejante al concepto de Data Access Object (DAO) [10].

- ✓ ***gui***: hace alusión a toda la parte con la interfaz gráfica. Excluye cualquier tipo de conexión a la base puesto que de eso se encargará el paquete *data* y más en concreto *data.entidad.* A esta forma de proceder separando lo gráfico de otra lógica (en este caso de conexión con un servidor) se le conoce como modelo vista – controlador [11].

- ✓ ***util***: los paquetes con la palabra *útil*, tanto dentro del paquete *kyb* como en *gui*, hacen referencia a utilidades. Dicho de otra manera, clases que van a ser utilizadas repetidas veces en distintos contextos. Por lo que es interesante agruparlas.

5. **Continuidad en el diseño estético de los menús**

Para comodidad de los operarios se intentará que los menús sean lo más parecidos posibles al software que ya poseen en la fábrica. Tomando como referencia fotografías obtenidas in situ.

6. **Algunos elementos de la interfaz serán plantillas**

Ciertos menús y componentes se replican en diferentes opciones dentro de la interfaz. Por lo que es sensato crear una plantilla gráfica cuyas características sean modificables.

Es el ejemplo de:

- ✓ Detalles de reglajes: creación de un nuevo reglaje, modificación en un reglaje existente, y mostrado de reglaje antes de lanzar un ensayo.
- ✓ Los paneles de las dos precargas y las catorce condiciones del reglaje.
- ✓ El diálogo selector de referencia para modificar un reglaje o para lanzar un ensayo.

3.3 Ejecución del programa

El programa puede ejecutarse de dos maneras.

Si se ejecuta por consola, es posible agregar a la línea un parámetro *--migracion*; en cuyo caso, correrá el método *main* de las clases del paquete de migración, antes de mostrar la interfaz.

Para que funcione correctamente y según lo esperado, es necesario que las dos bases (original y nueva) estén ya creadas en la máquina cliente y que la original tenga datos cargados.

Por ello, a la hora de instalar el software en el ordenador de la fábrica, deben ejecutarse los scripts mencionados en el apartado 2.4: Creación de la base de datos nueva.

Si no se escribe el parámetro, el programa lanza directamente la interfaz.

En el caso de que el servidor no estuviera funcionando, o la base nueva estuviese vacía, no funcionará correctamente. Los desplegables estarán vacíos y habrá errores en aquellas acciones que requieran conexión con la base.

El diagrama de funcionamiento normal de la interfaz sería como se muestra a continuación:

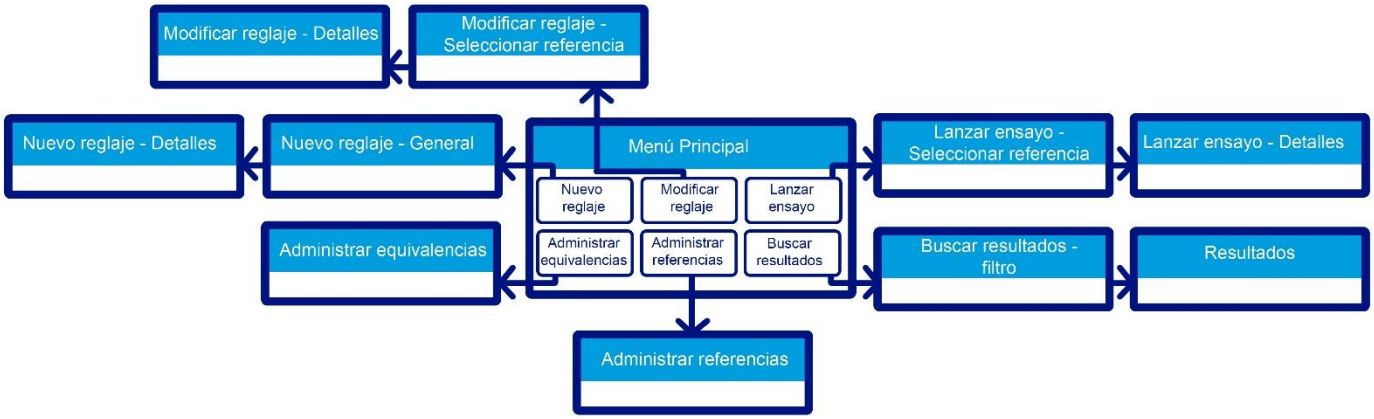


Ilustración 37: Diagrama de funcionamiento de la interfaz

3.4 Paquete de clases de entidades

Ya se mencionó anteriormente que se pretendía seguir el modelo vista – controlador, de tal forma que las conexiones y operaciones con la base, quedasen aparcadas del diseño y comportamiento gráfico de la interfaz.

El paquete *entidad* del proyecto agrupa unas clases que responden al esquema de la base de datos nueva y sus tablas.

Cada clase entidad representa a una tabla, y cada atributo supone una variable de objeto. Es necesario conocer qué métodos tienen las clases porque más adelante la interfaz los usa en las operaciones hacia la base de datos.

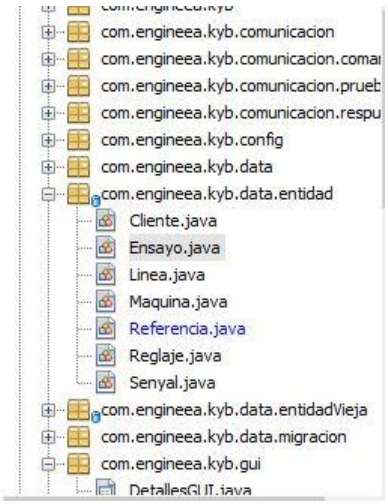


Ilustración 38: paquete de entidad y sus clases

Primero se describirán los atributos de cada clase y sus tipos. Después los métodos de la clase, con sus parámetros de entrada y sus datos de salida.

1. Referencia

Variables:

id (tipo Long)
nombre (String)
borrado (boolean)

Métodos:

- *Getters: getNombre(), getId(), getBorrado()* : cada uno de estos métodos, comúnmente conocidos como "getters", devuelve la variable del objeto especificada. Su nombre (tipo String), su id (Long) o su campo "borrado" (boolean) respectivamente.

- *getReferencias():* se conecta con la base y selecciona todas las referencias no borradas de la tabla *referencia*. Crea los objetos con sus variables pertinentes (nombre, id y borrado) y los devuelve en una lista.

- *getReferenciasConReglaje():* selecciona de una unión de las tablas *referencia, equivalencia* y *reglaje*, solamente aquellas en las que se encuentre reglaje asociado y con valor "borrado" (de las tres tablas) igual a 0. Devuelve una lista de objetos tipo Referencia.

- *getReferenciasSinReglaje*(): efectúa la acción complementaria al anterior, seleccionando referencias sin reglaje asociado o con valor "borrado" (en tabla *reglaje*) igual a 1. Las añade a una lista de objetos Referencia que devuelve.

- *getReferencia(Long id)* o *getReferencia(String nombre):* busca una referencia por id o por nombre (sin importar si está borrada o no) en la tabla *referencia* y la devuelve como un objeto Referencia.

- *tieneReglaje(String nombre):* devuelve un booleano en función de si la referencia, buscada por su nombre, tiene un reglaje asociado o no.

- *getNombreEquivalencia(String nombre):* devuelve una cadena de texto con el nombre de la equivalencia a la que pertenece una referencia, buscándola por su nombre en una unión de las tablas *referencia* y *equivalencia*.

- *getEquivalenciaIdporReferencia(String nombre):* devuelve el id de la equivalencia a la que pertenece una referencia (campo "equivalencia_id"), buscándola por su nombre en la tabla *referencia*.

- *getEquivalenciaId*(String equivalencia): devuelve el id de la equivalencia, buscándola por nombre en la tabla *equivalencia*. Sólo devuelve aquellas que no están borradas.

- *getEquivalencias*(): devuelve una lista de Strings con los nombres de equivalencias en la tabla *equivalencia* no borradas.

- *getReferencias*(): devuelve una lista de Strings con los nombres de referencias no borradas de la tabla *referencia*.

- *getReferenciasUnicas*(): devuelve una lista de Strings con los nombres de referencias de una vista creada en la base llamada *referenciasUnicas*.

  Una vista es una consulta guardada en la base de datos que facilita la escritura de la consulta en el código. En lugar de hacer una consulta larga en Java, con uniones y cláusulas; se aparca ese trabajo para la base de datos.

  La vista *referenciasUnicas* sirve para obtener aquellos nombres de referencias que no están agrupadas en equivalencias. Para ello se agrupa por el campo "equivalencia_id" y se cuentan los componentes del grupo. Es necesario hacer una subconsulta (consulta dentro de otra consulta).

  Tiene las siguientes líneas en SQL:

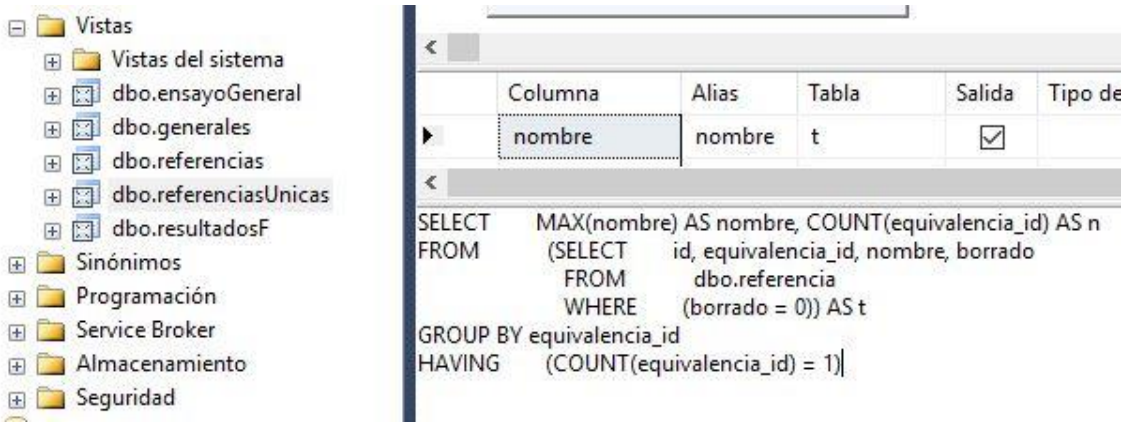


Ilustración 39: código SQL de la vista referenciasUnicas

- *nueva(String referencia)*: crea una equivalencia en la tabla *equivalencia* con el nombre introducido, y tras esto, una referencia en *referencia* con mismo nombre y el identificador de la equivalencia en "equivalencia_id". Ambas tablas con el campo borrado igual a 0. Devuelve un boolean verdadero si la operación ha sido exitosa.

- *crearEquivalenciaUnica(String equivalencia)*: este método está pensado para la situación en que una referencia se elimine de un grupo. Primero se crea una equivalencia en *equivalencia* con el nombre insertado. Después, se actualiza (Update en

SQL) el campo “equivalencia_id” en la tabla *referencia* de la referencia que corresponde. Devuelve un boolean verdadero si la operación ha sido exitosa.

- *borrar(String referencia)*: borra una referencia única. (La interfaz más adelante no permitirá borrar referencias que existen en un grupo).
  Actualiza el campo “borrado” a valor 1 de la referencia, buscada por su nombre. Después repite la operación en la tabla *equivalencia*. Debe hacerse en este orden para evitar errores de integridad referencial.

- *perteneceAGrupo(String referencia)*: busca si la referencia introducida está en la vista *referenciasUnicas*. Si está, devuelve un valor booleano falso (no pertenece a ningún grupo). Si no lo está, devuelve verdadero (sí pertenece a un grupo).

- *crearEquivalencia(String nombre, List<String> referencias)*: crea una equivalencia dada una lista de nombres de referencias. La equivalencia se crea en la tabla *equivalencia* con el formato de nombre: referencia1/referencia2/referencia3…

  Se marcan como borradas las equivalencias únicas correspondidas a las referencias de la lista, si las tuvieran, y se actualiza el campo “equivalencia_id” de cada referencia en la tabla *referencia*.

2. Reglaje

La clase *Reglaje* tiene una particularidad: dentro contiene dos clases estáticas Condicion y Precarga. Esto facilita la creación y relación de reglajes con sus condiciones y precargas.

Variables de *Reglaje*:

id (long)
equivalencia_id (long)
cliente_id (long)
revision (String)
modelo (String)
electronico (boolean)
serie (boolean)
linea_id (long)
fecha (Date)
gas (boolean)
borrado (boolean)
reglaje_id (long)
version (int)
condiciones (List<Condicion>, una lista de objetos Condicion)
precargas (List<Precarga>, una lista de objetos Precarga)

Variables de clase estática interna *Precarga*:

id (long)
ciclos (int)
cicloCaptura (int)
senyal_id (long)
amplitud (double)
frecuencia (double)
velocidad (double)
offset (double)

Variables de clase estática interna *Condicion*:

id (long)
ciclos (int)
cicloCaptura (int)
senyal_id (long)
amplitud (double)
frecuencia (double)
velocidad (double)
offset (double)

limiteSupCompresion (double)
limiteInfCompresion (double)
limiteSupTraccion (double)
limiteInfTraccion (double)

Métodos:

- Getters de las variables

- *getReglaje*(*long id*): busca un reglaje por su id introducida, establece los valores de sus atributos, y lo devuelve como objeto Reglaje.

- *getReglajePorEquivalenciaId*(*long equivalencia_id*): busca un reglaje no borrado que corresponda a la equivalencia especificada. Se selecciona el id del reglaje y se emplea el método anterior para poder devolver un objeto Reglaje.

- *borrar*(): actualiza el valor de campo borrado a 1 del objeto Reglaje al que se le aplica el método. Devuelve un booleano verdadero si la operación ha sido exitosa y falso, si ha habido errores.

- *nuevo(…)*: los parámetros de entrada de este método son los valores para todas las variables de *Reglaje,* incluidas las listas de precargas y condiciones. Si de esos

parámetros *reglaje_id* es nulo, significa que se está creando un reglaje nuevo. De tal forma que el campo "reglaje_id" de la base se relleneará gracias al trigger.

Si no es nulo, significa que se trata de un reglaje que se está modificando. Se insertará el mismo valor incrementado en uno.
Dentro del método se llama a otras dos funciones: *insertarPrecarga()* e *insertarCondicion()* . Los parámetros de entrada de las funciones son: el identificador del reglaje recién creado y la lista (de precargas, o de condiciones respectivamente).
Recorren la lista introduciendo en las tablas correspondientes una fila cada vez.

Se devuelve un objeto tipo Reglaje si todas las operaciones han sido exitosas. Si hay algún error, el objeto estará vacío (null).

- *insertarPrecarga(long reglaje_id, List<Object[]> precargas)*: es un método llamado desde el anterior para insertar en la tabla *precarga* los datos que corresponden. Se introducen las precargas como una lista de objetos Object. Esto es así porque no existe una clase específica de precarga (fuera de la clase Reglaje). Cada Object de esta lista, es un array con algunas variables numéricas: los atributos de la tabla *precarga*.

- *insertarCondicion(long reglaje_id, List<Object[]> condiciones)*: tiene el mismo funcionamiento que *insertarPrecarga()* pero referido a la tabla *condicion* y con las variables propias de los atributos en dicha tabla. Antes de ejecutar la entrada de datos en la base, se ordenan las condiciones por su velocidad con el método *ordenarcondicioensPorVelocidad()*.

- *ordenarCondicionesPorVelocidad(Object[] condiciones)*: utiliza un comparador y la clase Collections para ordenar las condiciones según su velocidad en orden ascendente. Dentro se sobreescribe (override) el método de comparación (compare) para especificar la forma de ordenar.

3. Ensayo

Como reglaje, Ensayo tiene una clase estática interna llamada Resultado.

Variables de *Ensayo:*

id (long)
reglaje_id (long)
referencia_id (long)
maquina_id (long)
numAmortiguador (String)
fecha (Date)
gas (double)
defectuoso (boolean)
comienzoCaptura (int)

resultados (List<Resultado>)
turno (int)

Variables de clase estática interna *Resultado*:

id (long)
cargaCompresion (double)
cargaTraccion (double)
velocidad (double)
defectuoso (boolean)

Métodos:

- Getters de las variables

- *getEnsayo*(*long id*): busca un ensayo por su id introducida, establece los valores de sus atributos, y lo devuelve como objeto Ensayo.

- *calcularTurno(Date fecha)*: en la fábrica se siguen tres turnos dependiendo de las horas: turno 1 de 6h a 14h, turno 2 de 14h a 22h y turno 3 de 22h a 6h. Este método devuelve un número entero correspondiente al turno dependiendo de la hora que indica el dato fecha.
  Esta variable de turnos no existe como atributo en la base de datos puesto que es fácilmente calculable a través de la fecha del ensayo; sin embargo es necesaria ya que es el modo de trabajo en la fábrica, y la presentación de resultados en el software actual.

- *getResultados(long ensayo_id)*: dado el identificador de un ensayo, busca los registros de resultados asociados en la tabla *resultado*. Crea objetos tipo Resultado, con sus variables especificadas y devuelve una lista de éstos.

- *getEnsayos(String fecha de inicio, String fecha de fin, long referencia_id)* o *getEnsayos(String fecha de inicio, String fecha de fin)*: se pueden buscar ensayos especificando un intervalo de fechas (inicio y fin) solamente, o por fechas y de una referencia en concreto. Los dos métodos devuelven una lista de objetos Ensayo con los resultados de la búsqueda.

4. Cliente

Variables:

id (tipo Long), nombre (String)

Métodos:

- Getters de las variables

- *getClientes*() : devuelve una lista de objetos Cliente (esta misma clase). Se conecta con la base y selecciona todos los clientes de la tabla *cliente*. Crea un objeto con las variables id y nombre las que corresponda, y lo añade a la lista.

- *getCliente(Long id)* o *getCliente(String nombre)*: devuelven un objeto Cliente, y se puede buscar tanto por id como por nombre.

- *crearCliente(String nombre)*: inserta en la tabla *Cliente* el registro con el nombre que se le pasa por parámetro. Devuelve el identificador en tipo Long.

5. Maquina

Igual funcionamiento que la clase *cliente*, pero con la tabla *maquina*. No tiene un método para crear máquinas.

6. Senyal

Igual funcionamiento que las clases anteriores, pero con la tabla *senyal*. No tiene un método para crear señales.

7. Linea

Igual funcionamiento que las clases anteriores, pero con la tabla *linea*. No tiene un método para crear líneas.

3.5 Menú principal

Al ejecutar el programa lo primero que se muestra es un menú principal con seis botones, como se presenta en la Ilustración 40, ocupando el espacio en porciones iguales. De título: operaciones.

Se añaden unos iconos simples a los botones desde la interfaz de diseño de Netbeans, dibujados a partir de imágenes de libre distribución en internet [12].

Se prefiere el formato .png de imagen para poder aprovechar su característica de transparencia.

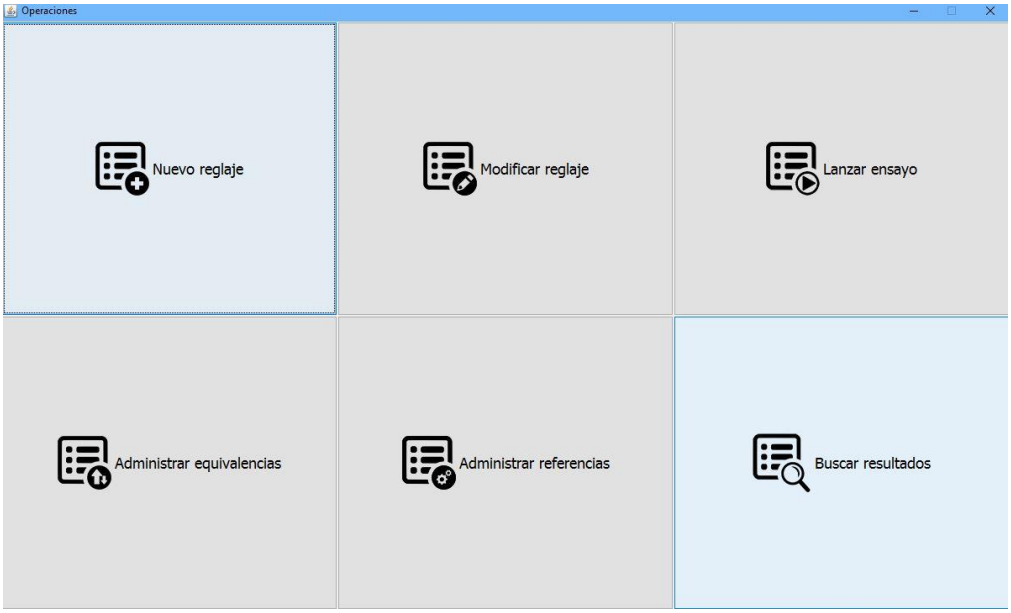


Ilustración 40: Menú principal

La lógica del menú principal es simple: cada botón responde al ser clickado y realiza una acción. El evento capturado (Java) en cada botón es: *actionPerformed*.

Hay seis posibles opciones, dependiendo del botón clickado se creará el JDialog del menú que corresponda.

Netbeans tiene un apartado de diseño en el cual se pueden arrastrar los componentes de una paleta hasta la interfaz, modificar con el ratón el tamaño de la ventana que se está diseñando, etc.

Gracias a este apartado, se han colocado los elementos en la interfaz, y Netbeans se ocupa de generar código automático.

Todo lo relativo al comportamiento de los gráficos se puede definir tanto con el apartado de diseño del IDE o escribiendo código. Es el caso de los eventos capturados para cada componente, o la imposición de sus características (el título, el tamaño, el texto interior, etc.).

3.6 Menú *Nuevo reglaje*

3.6.1 Nuevo reglaje - General

Desde el menú principal, el botón en la esquina superior izquierda accede a la creación de un nuevo reglaje; de título: Nuevo reglaje – General.

En los diálogos se ha seguido la siguiente convención para los títulos:

Botón accedido desde el menú principal – función

Desde este menú se puede: o bien añadir un reglaje a una referencia que no tenga; o bien crear una referencia no existente y su reglaje de manera conjunta.

Ya hay un menú en la interfaz para crear referencias, sin embargo, se deja la opción disponible puesto que, según el cliente, es una operación muy común en la fábrica y es la manera de trabajo hasta la fecha.

Se ha diseñado el menú con las fotografías tomadas en la fábrica como referencia:

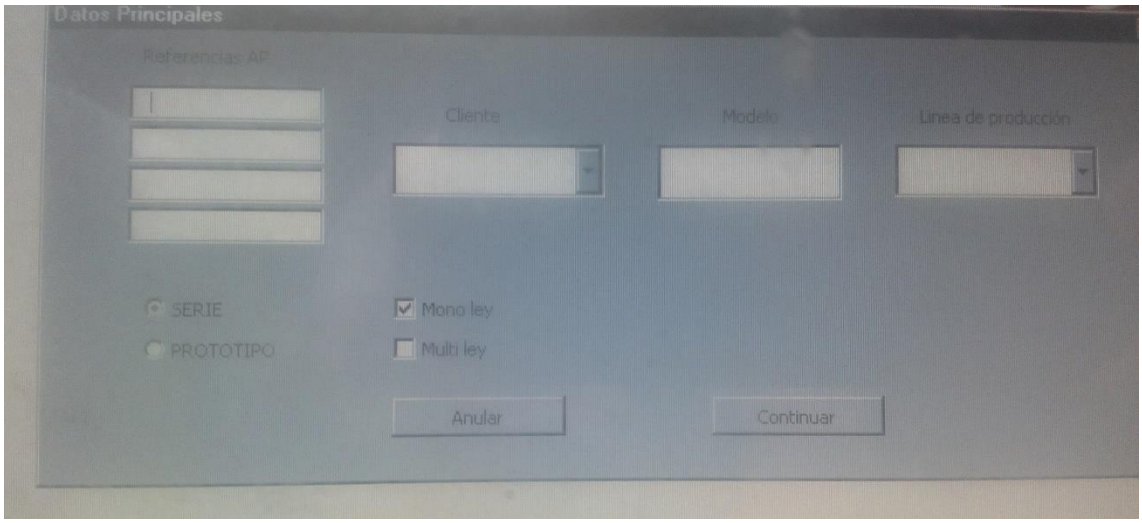


Ilustración 41:fotografía del software actual de la fábrica con un menú de creación de reglaje

Se advierte que en el software original se introducen las referencias de una vez a mano para crear equivalencias. No resulta muy práctico si se quieren crear equivalencias de más de cuatro referencias. Por ello se decidió aislar y gestionarlo en un apartado distinto. Véase el apartado 3.9: Menú Administrar equivalencias.

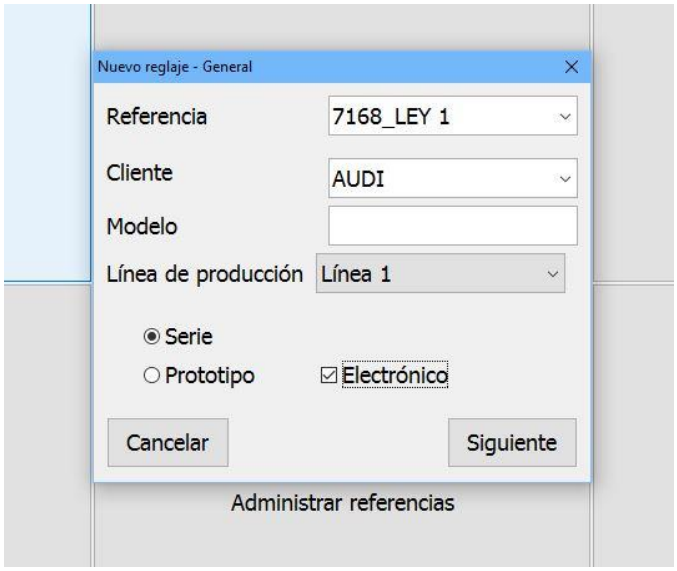


Ilustración 42: Nuevo reglaje - General

El menú tiene una lista desplegable (combobox) con las referencias de la base de datos sin reglajes asociados.

Es editable por dos razones: una, para que sea posible crear una nueva referencia a la par que su reglaje. Y otra, por comodidad de búsqueda. Conforme se escribe en el campo, la caja de texto tiene autocompletado. Este tipo de lista desplegable se usará en muchos apartados.

La segunda lista despegable contiene los clientes de la base de datos. Es editable y permite más adelante crear un cliente nuevo en la base.

El menú tiene una caja de texto simple para escribir el modelo del amortiguador.

Lína de producción es una lista desplegable con las líneas de producción de la fábrica.

En el inicio de la ejecución del diálogo, se cargan las listas de referencias, clientes y líneas de producción. Gracias a los métodos del paquete entidad:

- método *getReferenciasSinReglaje()* de la clase *Referencia*
- método *getLineas()* de la clase *Linea*
- método *getClientes()* de la clase *Cliente*

El recuadro (checkbox) con nombre “Electrónico” define si el amortiguador tiene un sistema de suspensión electrónica (si está marcado) o no (desmarcado).

Los botones de selección (radiobutton) de “Serie” y “Prototipo” describen si el amortiguador se fabrica en serie o si es un prototipo individual. Seleccionar uno desmarca el otro.

El funcionamiento de este menú es el siguiente:

- ✓ Se accede aquí desde el botón del menú principal.

- ✓ Al clickar el botón de cancelar, se cierra esta ventana, dejando plenamente visible el menú principal.

- ✓ Al clickar el botón de siguiente, se hacen una serie de comprobaciones antes de cambiar al siguiente menú. Estas son:

1. **La referencia escrita contiene una barra “/”.**

No se pueden crear reglajes directamente para equivalencias. Se hace para referencias únicas que no tengan reglaje asociado o no existan en la base.

Si la referencia contiene una barra, salta un mensaje informando de ello. Tras aceptar, continúa con las operaciones.

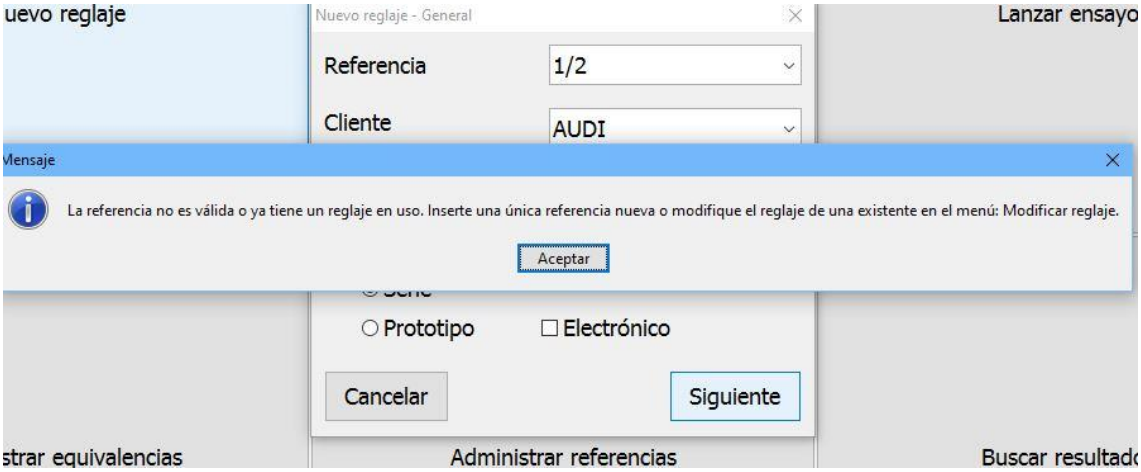


Ilustración 43: mensaje de error al introducir barras.

2. **La referencia escrita o seleccionada, tiene un reglaje asociado y está en uso.**

Es decir, que el campo "borrado" de la referencia existe y el valor de su campo "borrado" es 0.

Si se escribe una referencia existente que no aparece en la lista desplegable, se comprueba que no tiene un reglaje con el método *tieneReglaje()* de la clase entidad Referencia.

Aparecerá el mismo mensaje de la Ilustración 43: mensaje de error al introducir barras..

3. **El cliente está vacío o el texto en cliente es demasiado largo**

El cliente debe ser o bien uno seleccionado de la lista, o uno nuevo. Pero dicho cliente nuevo, no puede superar un tamaño de 300 caracteres. Se aprovecha la comprobación para evitar

errores desde la base (ya que en la base se define un campo de texto para “cliente” inferior a 300 caracteres).

Si el cliente es nuevo, aparece un mensaje que pregunta al usuario si desea crear el cliente. Seleccionar “No”, devolverá a la pantalla de Nuevo reglaje – General para seleccionar un cliente existente. Seleccionar sí, pasará a la siguiente pantalla.

Se aprovecha la creación de cliente en este apartado puesto que el campo es editable, y es el único modo de hacerlo en la interfaz.

Ilustración 44: Mensaje de interrogación para crear un cliente nuevo

4. El modelo está vacío o el texto en modelo es demasiado largo

Comportamiento parecido al campo cliente.

Cuando un error se da haciendo operaciones en la base de datos, la interfaz deja de funcionar como se espera. Por ello se insiste mucho en el control de errores, evitando trasladarlos a la base. Así, se intenta anteponer a estos eventos programando desde Java la interfaz.

- ✓ Si todas las comprobaciones son correctas, se procede a la siguiente pantalla.

Se crea una instancia de la clase de Detalles, se le pasan ciertos parámetros y se oculta el diálogo actual.

Los parámetros son: este mismo diálogo, para poder acceder a lo escrito en sus campos y lo seleccionado; y si se trata de lanzar un ensayo o no.

Este último parámetro es necesario puesto que la pantalla a continuación se trata de una plantilla empleada en varios diálogos.

Además, se incluye una variable booleana que describe si se va a crear un cliente nuevo.

3.6.2 Nuevo reglaje - Detalles

Nuevo reglaje - Detalles se trata de clase que funciona como una plantilla común a tres situaciones distintas.

Por ello, el JDialog tiene dos constructores distintos: uno para nuevo reglaje, y otro para modificar reglajes y lanzar ensayos.

Se diseña así puesto que para crear el nuevo reglaje se necesitan varios datos de la pantalla anterior, mientras que modificar y lanzar ensayos sólo requerirá el nombre de una referencia.

La apariencia de este diálogo es acorde a la siguiente fotografía tomada en la fábrica:

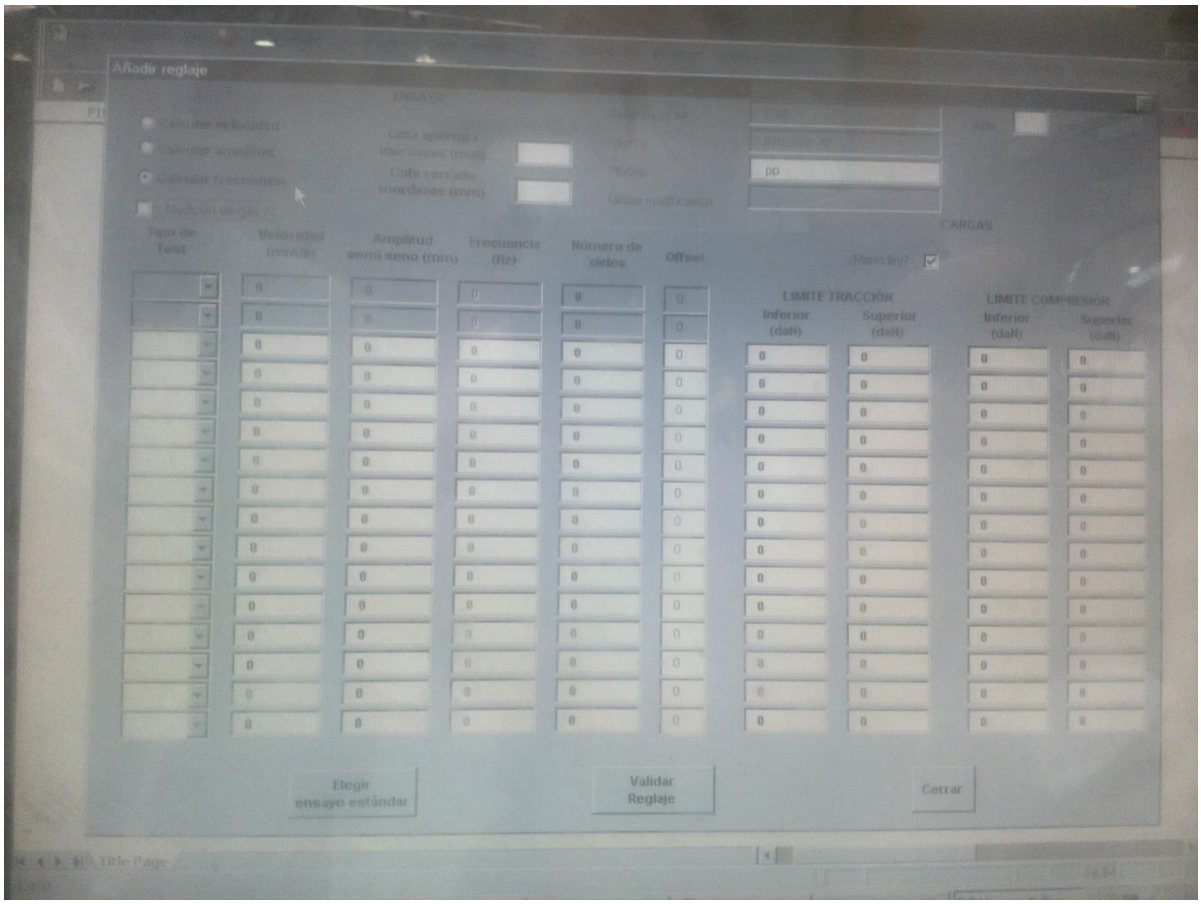


Ilustración 45: fotografía del software actual de la fábrica con la pantalla de detalles del reglaje

Se sigue el diseño de color gris para señalar las precargas.

Algunos elementos no se incorporan puesto que ya no tienen uso. Como son, por ejemplo, la ejecución de un ensayo en dos máquinas a la vez, la distinción entre monoley o multiley, o las mordazas (las máquinas de la fábrica no poseen mordazas).

En las siguientes ilustraciones, primero se introducen una serie de datos generales para el nuevo reglaje en el menú Nuevo reglaje - General, y después se muestra el diálogo de detalles.

Ilustración 46: ejemplo de introducción de datos en Nuego reglaje – General

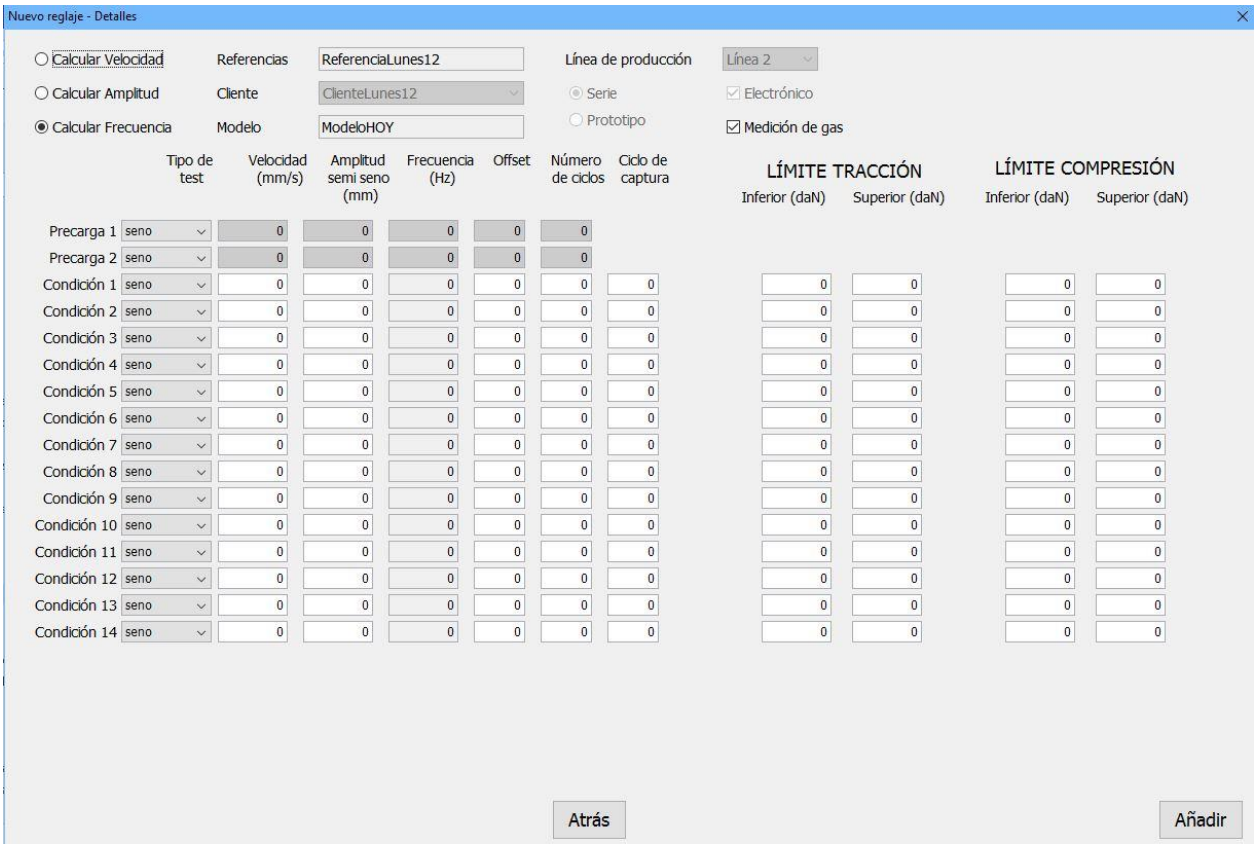


Ilustración 47: interfaz Nuevo reglaje – Detalles

En este diálogo no se puede aplicar el criterio de grandes fuentes o elementos debido a la gran cantidad de componentes.

La plantilla varía en las tres situaciones en: la posibilidad de editar/habilitar ciertos campos, la visibilidad de campos, botones, etiquetas y los títulos.

En la parte superior de Nuevo reglaje - Detalles aparecen los datos generales introducidos en el menú anterior, pero no son editables.

Sí lo son, el resto de campos, excepto la columna a calcular: seleccionada desde los tres radiobutton en la esquina superior izquierda (calcular velocidad/amplitud/frecuencia).

Se calcula su resultado numérico dependiendo de la señal escogida y su fórmula.

Este comportamiento también aparecerá en el menú Modificar reglaje – Detalles.

Por eficiencia, las líneas de precarga y condición, son paneles que se generan dinámicamente al correr el programa. Así se evita el tener que manejar muchos elementos individualmente en la interfaz.

Las filas son iguales, a excepción del texto en la etiqueta de cada uno ("condición 1, condición 2…condición 14"). Precarga y condición existen como panel individual en el paquete: *gui.util*. Debido a que ambas tienen características iguales, extienden de una clase llamada *línea*, que contiene algunos métodos comunes. Por ejemplo, el poder parsear (hacer un cambio de formato), entre tipos de datos. Cadena de texto (String) a número decimal (double), a número entero (int), etc.

Ilustración 48: panel condicion en el paquete de utilidades de la GUI

El panel contiene la etiqueta, un selector para el tipo de señal (combobox), y cajas de texto.

Al introducir cada panel, se cargan los tipos de señales como elementos de la lista desplegable. Se obtienen gracias al método *getSenyales()* de la clase *Senyal* en el paquete de entidad.

En un primer acercamiento, el cálculo de columnas se pensó que debía capturarse con el evento *keyReleased.* Es decir, cuando se suelta una tecla en un campo de escritura, calcularía el número.

Esto actualizaba el resultado en el campo elegido, conforme se escribían los números en cualquiera de los otros dos.

Sin embargo, daba lugar a otros inconvenientes: no se podía mover el cursor mediante las teclas de flecha por los caracteres ya que disparaba el evento de tecla liberada y el cursor volvía al inicio del campo.

También se detenía y generaba error si la cadena de texto no podía convertirse a número en un instante dado.

Se prefiere entonces esperar a introducir todo el valor y hacer el cálculo una vez que se pierde el foco del mismo (evento *focusLost*). Se pierde el foco de un elemento, por ejemplo, haciendo click en otro elemento distinto.

La contraparte es que no se calcula en tiempo real conforme se escribe.

El funcionamiento normal del cálculo de columnas es el que sigue.

1) Se comprueba que el texto introducido es un número correcto.
   En todos los campos se aceptan números decimales, excepto en los referidos a ciclos, donde sólo se aceptan enteros positivos. Se intenta traducir la cadena de texto a número decimal (tipo double) o a entero (int) según corresponda; y si no es posible se muestra un mensaje informativo del error. Por ejemplo, si se introdujesen caracteres del alfabeto.

2) Si el número es admitido, se calcula el resultado del campo elegido (frecuencia, amplitud o velocidad) y en función del tipo de señal seleccionado en la lista, ya que tienen diferente fórmula.

3) A la par que este cálculo, se aprovecha para cambiar una coma decimal por punto, si la hubiere. La forma de hacerlo es forzando la lengua inglesa al mostrar el resultado numérico.
   También se incluye una condición para no mostrar posiciones decimales si el número es entero, y tres posiciones como máximo si no lo es.

```
protected void cambiarCifraDecimalCampo(JTextField campo){
    Double valor = getDouble(campo);
    String result;
    if(valor % 1 < 0.0001){
        result = String.format(Locale.ENGLISH,"%.0f", valor);
    }else{
        result = String.format(Locale.ENGLISH,"%.3f", valor);
    }
    campo.setText(result);
}
```

Los datos en la interfaz se muestran en unidades de mm, mm/s y daN para: amplitud, velocidad, y cargas.

El cliente postula que es preferible el sistema internacional, pero los operarios de la fábrica usan más estas unidades. Por lo que decide dejar de este modo.

Antes de colocar los valores de las condiciones, se ordenan gracias a un comparador. De tal forma que no importa el modo en que se escriban en la interfaz, en la base el valor de "numCondicion" (que determina el número de condición de reglaje) irá de menor a mayor con la velocidad de la condición.

```
private static void ordenarCondicionesPorVelocidad(List<Object[]> condiciones) {
    Collections.sort(condiciones, new Comparator<Object[]>() {
        @Override
        public int compare(Object[] o1, Object[] o2) {
            Double velocidad1 = (Double) o1[1];
            Double velocidad2 = (Double) o2[1];
            return velocidad1.compareTo(velocidad2);
        }
    });
}
```

Ilustración 49: método que ordena las condiciones por su velocidad en orden creciente

El funcionamiento de este menú es el siguiente:

- ✓ Se accede aquí desde el menú Nuevo reglaje – General.

- ✓ Al pulsar el botón “cancelar”, se cierra la ventana Nuevo reglaje – Detalles y se activa la visibilidad de la anterior: Nuevo reglaje – General.

- ✓ El botón “añadir” es un elemento de la plantilla que realiza diferentes acciones según el menú del que se trate: Nuevo reglaje, Modificar reglaje o Lanzar ensayo.
  Por lo que, primero, comprueba en qué caso se está trabajando. En este caso, es la creación de un nuevo reglaje. Antes de continuar, hace una serie de comprobaciones:

1. **Los límites de cargas son coherentes**

Los límites de las condiciones deben ser coherentes: los valores de límites inferiores deben ser menores que los límites superiores en todas las condiciones; para tracción y compresión.

Si la comprobación falla, muestra un mensaje de error informativo, y no crea el reglaje, ni la referencia o el cliente (en caso de cliente nuevo).

2. **Existe, al menos, un campo distinto de 0 en una condición**

No se crea un reglaje a no ser que al menos uno de los campos de una condición, tenga valor. Esto evitará que existan reglajes vacíos, sin condiciones ni precargas asociadas.

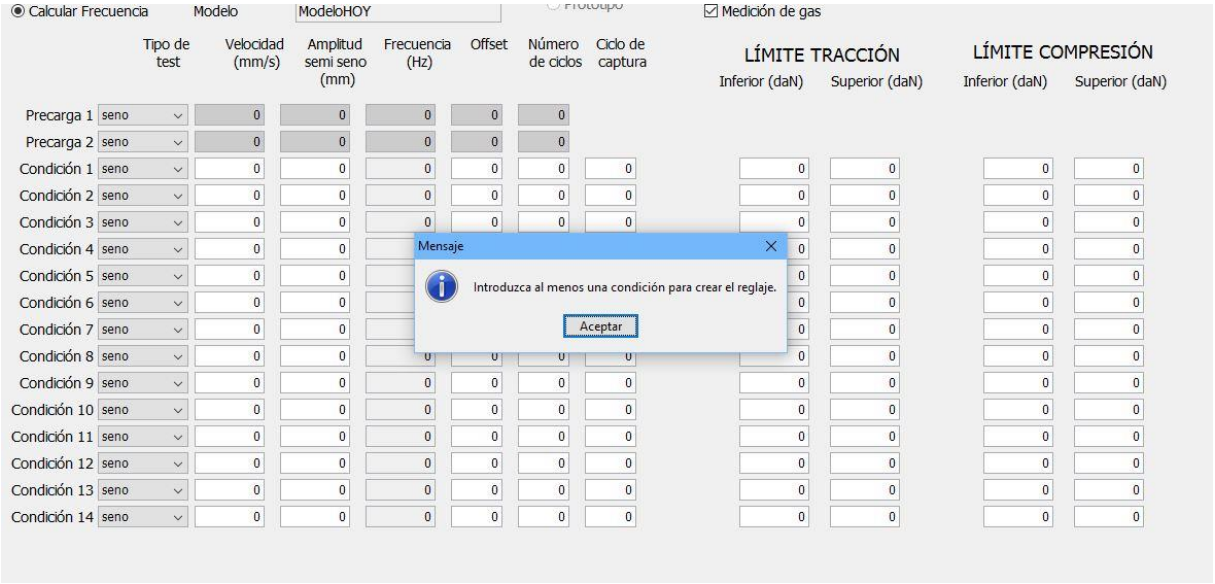


Ilustración 50: no se crea un reglaje si ninguna condición tiene valores numéricos

Si todos los datos son válidos, el procedimiento programado es:

1) Si se ha elegido crear un nuevo cliente en Nuevo reglaje - General, se crea un nuevo registro con su nombre en la tabla *cliente*, con el método *crearCliente()* de la clase *Cliente*.

2) Si la referencia no se encuentra en la lista de Nuevo reglaje - General, se crea en las tablas *equivalencia* y *referencia* con el método *nueva()* de la clase *Referencia*.

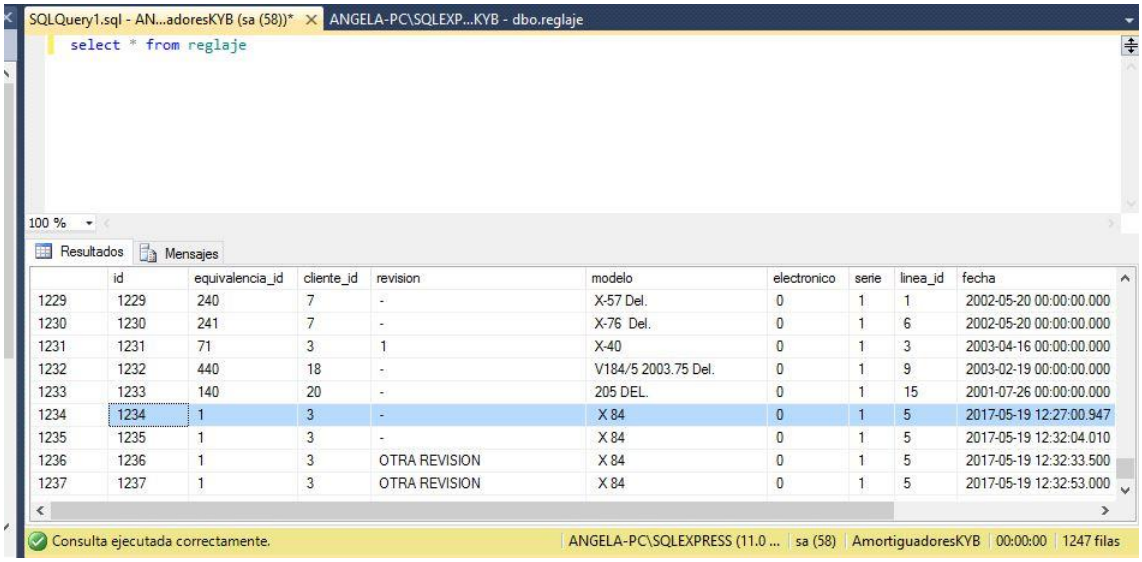
3) Si la referencia se encuentra en la lista, se guarda el identificador de su equivalencia con el método *getEquivalenciaIdporReferencia()*; para insertarlo más tarde en el reglaje.

4) Se guarda el identificador del cliente buscándolo en su tabla con el método *getCliente()*.

5) Se recogen los datos de la interfaz para crear el reglaje. Se recuerda el esquema de la base de datos en la Ilustración 5.

   Algunos datos se consiguen de la interfaz, mientras que otros se crean en el momento. Por ejemplo, el dato de fecha se gestiona desde la base de datos.
   El campo "version" alude al número de veces que se ha modificado un reglaje; al ser de nueva creación se le da valor 0 por defecto.
   El campo "revisión" se utiliza en posteriores modificaciones como anotación para los operarios. Por defecto para un nuevo reglaje se inserta un guión "– ".
   Se escoge este símbolo puesto que está muy presente en los datos del campo "Revision reglaje" de la base de datos original (tabla *Info_referencia*).

6) Se crea un registro en la tabla *reglaje* con los datos recogidos en el paso 5. Se usa el método *nuevo()* de la clase *Reglaje*. Este a su vez creará los registros en las tablas *precarga* y *condicion*.



	id	equivalencia_id	cliente_id	revision	modelo	electronico	serie	linea_id	fecha
1229	1229	240	7	-	X-57 Del.	0	1	1	2002-05-20 00:00:00.000
1230	1230	241	7	-	X-76 Del.	0	1	6	2002-05-20 00:00:00.000
1231	1231	71	3	1	X-40	0	1	3	2003-04-16 00:00:00.000
1232	1232	440	18	-	V184/5 2003.75 Del.	0	1	9	2003-02-19 00:00:00.000
1233	1233	140	20	-	205 DEL.	0	1	15	2001-07-26 00:00:00.000
1234	1234	1	3	-	X 84	0	1	5	2017-05-19 12:27:00.947
1235	1235	1	3	-	X 84	0	1	5	2017-05-19 12:32:04.010
1236	1236	1	3	OTRA REVISION	X 84	0	1	5	2017-05-19 12:32:33.500
1237	1237	1	3	OTRA REVISION	X 84	0	1	5	2017-05-19 12:32:53.000

Ilustración 51: ejemplo de registros en la tabla reglaje. Vista desde SQL Server Management Studio

✓ Si ocurre un fallo en la creación del reglaje se muestra un mensaje de error; de lo contrario se muestra un mensaje informativo del éxito de la operación y se oculta el diálogo. Volviendo a visualizar el menú principal.

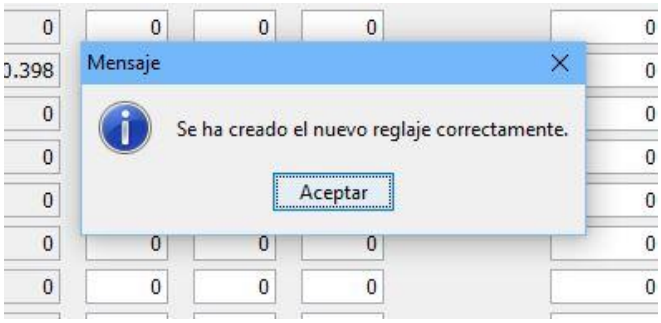


Ilustración 52: inserción correcta de reglaje

3.7 Menú Modificar reglaje

Desde el menú principal, el botón central superior accede a la modificación de reglajes.

Desde este menú se puede: eliminar un reglaje o modificarlo. Editando sus datos generales, de condiciones, precargas, etc.

3.7.1 Modificar reglaje – Seleccionar referencia

Semejante a las fotografías del software de la fábrica, aparece un selector de referencia en primer lugar.

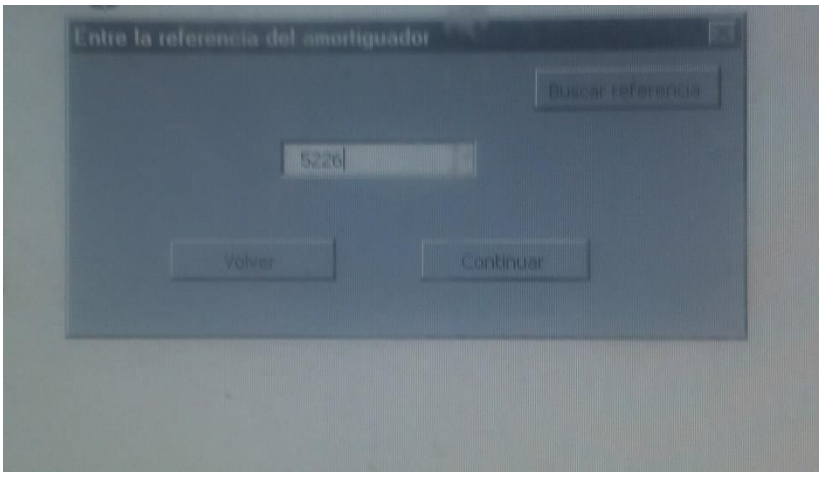


Ilustración 53: fotografía del software actual de la fábrica sobre la inserción de referencias

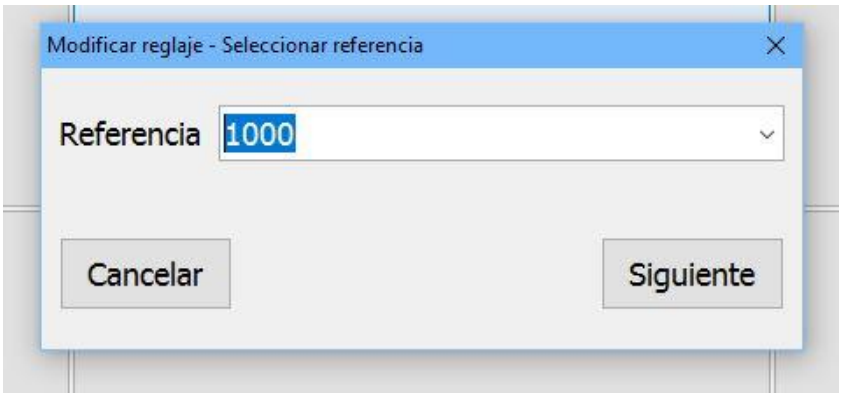


Ilustración 54: Modificar reglaje - Seleccionar referencia

El buscador es una de las plantillas definidas: se emplea al modificar un reglaje y antes de lanzar un ensayo.

El título del diálogo dependerá de qué opción de las dos se ha pulsado. Además, se envía un valor booleano verdadero, si se trata de un ensayo, y falso si se trata de la modificación de un reglaje. En base a este valor después del buscador se procederá a una pantalla distinta (ensayo o modificación de reglaje). Exceptuando esto, el funcionamiento del selector es idéntico en ambas situaciones.

La lista sirve para elegir una referencia válida y con reglaje en uso, por lo que muestra aquellas que cumplan las dos restricciones. Se cargan sus elementos gracias al método del paquete entidad de la clase Referencia: *getReferenciasConReglaje()*.

Los reglajes van asociados a equivalencias, pero parece más coherente buscar por cada referencia para evitar fallos en la búsqueda: es posible que un operario quiera buscar la equivalencia **1222/1333** y en la base está almacenada como 1111/**1222/1333**.

Buscar manualmente la equivalencia deseada sería costoso puesto que hay alrededor de 2000 referencias y 1000 equivalencias.

Es a causa de este alto número que se añade la opción de editar la caja de texto y el autocompletado.

El funcionamiento del selector de referencia es el siguiente:

- ✓ Se accede aquí desde el botón del Menú principal.
- ✓ Al clickar el botón “cancelar”, se cierra esta ventana, dejando visible el menú principal.
- ✓ Al clickar el botón “siguiente”, se hace una comprobación sobre la referencia. Esta es:

1. **La referencia escrita existe y tiene reglaje válido**

Se mostrará un mensaje informativo de error si la referencia escrita no aparece en la lista. Para que aparezca en la lista, la equivalencia a la que pertenece debe poseer un reglaje asociado con el campo “borrado” igual a 0. La referencia y la equivalencia tampoco pueden estar borradas.

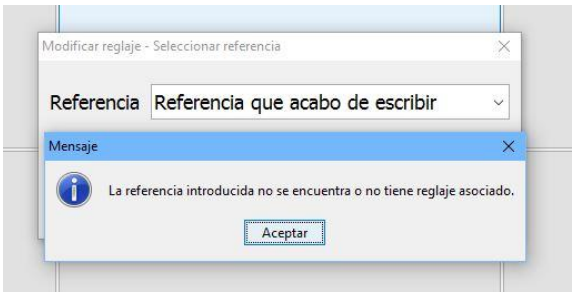


Ilustración 55: mensaje de error al buscar una referencia en el selector de Modificar reglaje

✓ Si la comprobación resulta correcta, se procede a la siguiente pantalla.
Se crea una instancia de la clase de Detalles con ciertos parámetros y se oculta el diálogo actual.
Los parámetros son: el nombre de la referencia seleccionada y un booleano que define si se trata de un ensayo o no.

3.7.2 Modificar reglaje – Detalles

Se usa la plantilla de detalles ya presentada en el apartado 3.6.2: Nuevo reglaje - Detalles con algunas diferencias.

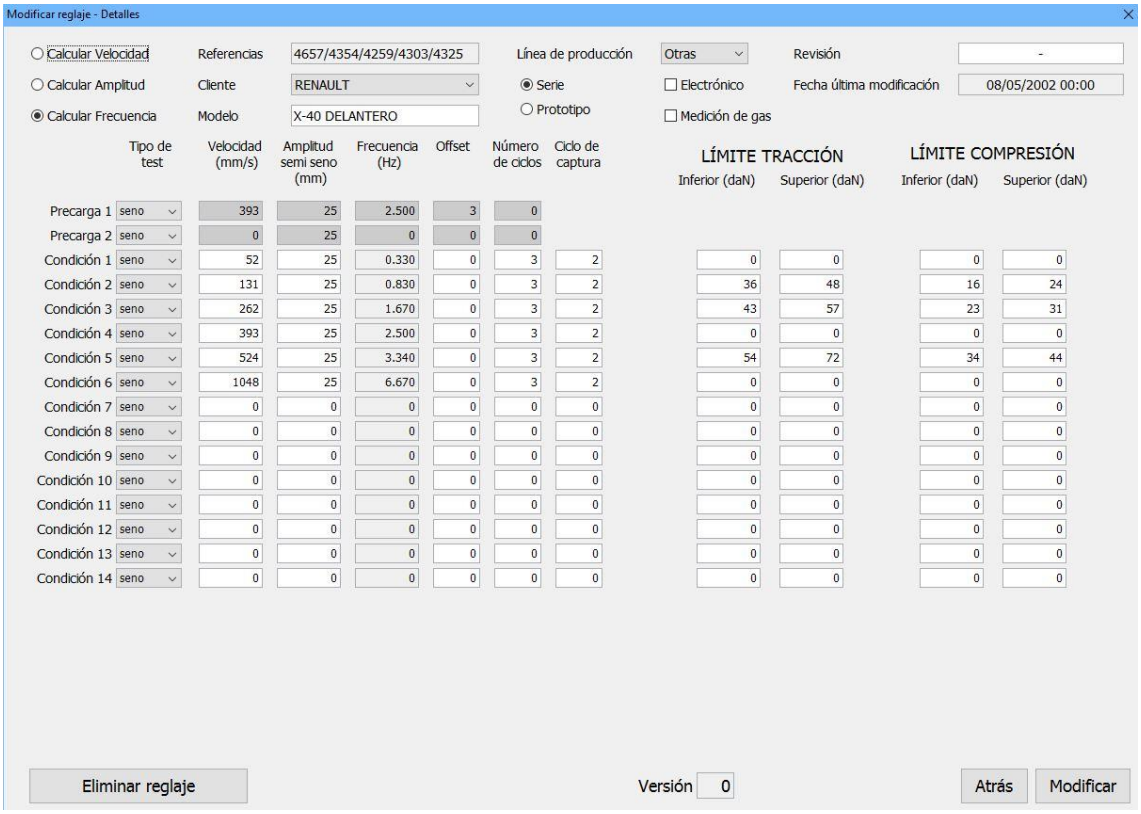


Ilustración 56: Modificar reglaje – Detalles

La obtención de los datos se hace del siguiente modo:

1) Se busca el identificador de la equivalencia a la que pertenece la referencia elegida. Se emplea el método *getEquivalenciaIdporReferencia()* de la clase *Referencia*. A partir de éste, se accede a su reglaje: *getReglajeporEquivalenciaId()* de la clase *Reglaje*.

2) Ya con el reglaje, se cargan los datos generales en los elementos del panel superior (variables de la clase *Reglaje*) y los detalles de precargas y condiciones en el resto del diálogo (variables de las clases internas de *Reglaje*).

La visualización del diálogo es la misma que en Nuevo reglaje - Detalles, pero se han añadido atributos como: la versión, fecha, y revisión del reglaje.

Se elige mostrar estos campos para mantener informado al operario de los cambios en un reglaje, aunque algunos (como la versión), son prescindibles. La fecha supone la última vez que un reglaje fue modificado, y el número de la versión, cuántas veces lo ha sido.

La revisión es un campo de texto puramente informativo de modificaciones hechas sobre un reglaje.

Además de cambiar el título en este uso del a plantilla; cambia el texto en el botón de “añadir” (ahora “modificar”), y aparece un botón más, “eliminar reglaje”.

Estos cambios se gestionan alternando la visibilidad de los componentes.

A excepción de la equivalencia, versión y fecha, el resto de la información se puede modificar. Por ello gran parte de los componentes son editables y seleccionables. El cargado de datos en las listas (como en el tipo de señal o los clientes), funciona del mismo modo que en menús anteriormente mencionados.

La razón por la cual la equivalencia no es editable, es que este menú no es el adecuado para ello. La fecha y la versión no se editan porque deben ser campos automáticos.

La generación de paneles para precargas y condiciones, y la lógica en el cálculo de las columnas se mantiene igual.

Una incidencia que ha surgido durante el desarrollo de la interfaz, ha sido ver que algunos reglajes guardados no cumplían las fórmulas de sus condiciones. Esto es, al clickar en un campo y perder el foco, la columna calculada cambiaba su resultado a un valor distinto al que había guardado.

Era lógico pensar que podía ser un fallo debido a la precisión, y ocurría en la mayoría, pero hubo algunas excepciones donde el número era claramente distinto. Esto llevó a consultar con el cliente, quien ofreció la segunda fórmula para las señales tipo rampa. Aún y todo, hubo reglajes que no cumplían. Se debe, con seguridad, a pruebas realizadas sobre los datos.

Otra incidencia de interés, fue comprobar que muchos reglajes (los más antiguos en fecha), no albergaban la hora. Se mostraba: 00:00:00. Ocurrió que algunos sí que contenían la hora (los de últimos años); pero no terminaba de mostrarse correctamente en la interfaz.

La causa es que al captar el dato de la base para tratarlo en Java, debe hacerse como si la columna fuese un Timestamp, y no una fecha Date (*conjuntoDeResultados.getTimestamp().* Ya que las fechas no almacenan hora.

Aunque después sí se puede guardar en una variable Date, del paquete *util* de Java sin perder esa información.

fecha es una variable de la clase Reglaje de tipo Date:

```
reglaje.electronico = rs.getBoolean("electronico");
reglaje.serie = rs.getBoolean("serie");
reglaje.linea_id = rs.getLong("linea_id");
reglaje.fecha = rs.getTimestamp("fecha");
reglaje.gas = rs.getBoolean("gas");
reglaje.borrado = rs.getBoolean("borrado");
reglaje.reglaje_id = rs.getLong("reglaje_id");
reglaje.version = rs.getInt("version");

sql = "SELECT * FROM condicion WHERE reglaje_id = ? ORDER BY numCondicion ASC";
statement = Conexion.Conexion.prepareStatement(sql);
statement.setLong(1, id);
rs = statement.executeQuery();
```

El funcionamiento de este menú es el siguiente:

- ✓ Se accede aquí desde la selección de referencia en Modificar reglaje – Seleccionar referencia

- ✓ Al pulsar el botón "cancelar", se cierra la ventana Modificar reglaje – Detalles y se activa la visibilidad de la anterior: Modificar reglaje – Seleccionar referencia.

- ✓ Al pulsar el botón "eliminar reglaje", suceden las siguientes acciones:

1) Se muestra un JOptionPane de mensaje interrogador. Pregunta al usuario si está seguro de que quiere eliminar el reglaje. Si se pulsa "No", el mensaje se cierra y se sigue visualizando la pantalla Modificar reglaje – Detalles sin realizar ninguna acción. Si se pulsa "Sí" se procede a borrar el reglaje.

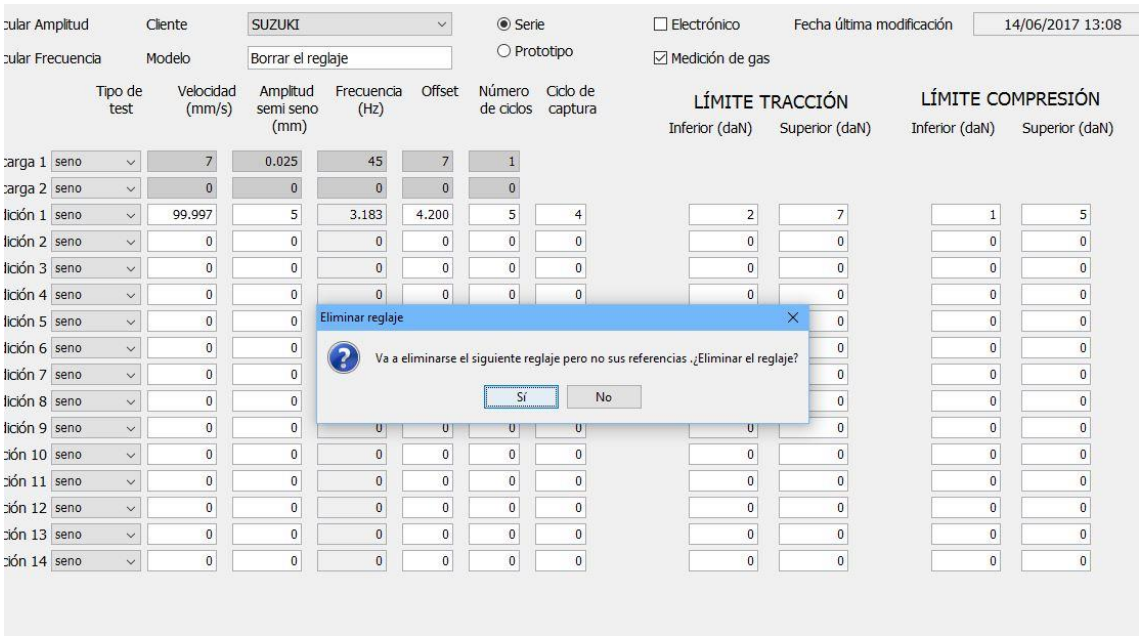


Ilustración 57: Mensaje de interrogación antes de eliminar un reglaje

2) Para borrar el reglaje, se busca el identificador del reglaje mediante el de la equivalencia, gracias al método *getReglajePorEquivalenciaId*() de la clase *Reglaje*.

3) El método del paso anterior devuelve un objeto Reglaje, y sobre él se usa el método de la clase *borrar();* que ya se escribió: actualiza el campo "borrado" a valor 1 en la base de datos.

4) Si la operación ha fallado en algún punto, se muestra un mensaje de error. Si ha borrado correctamente el reglaje, así lo informa también.

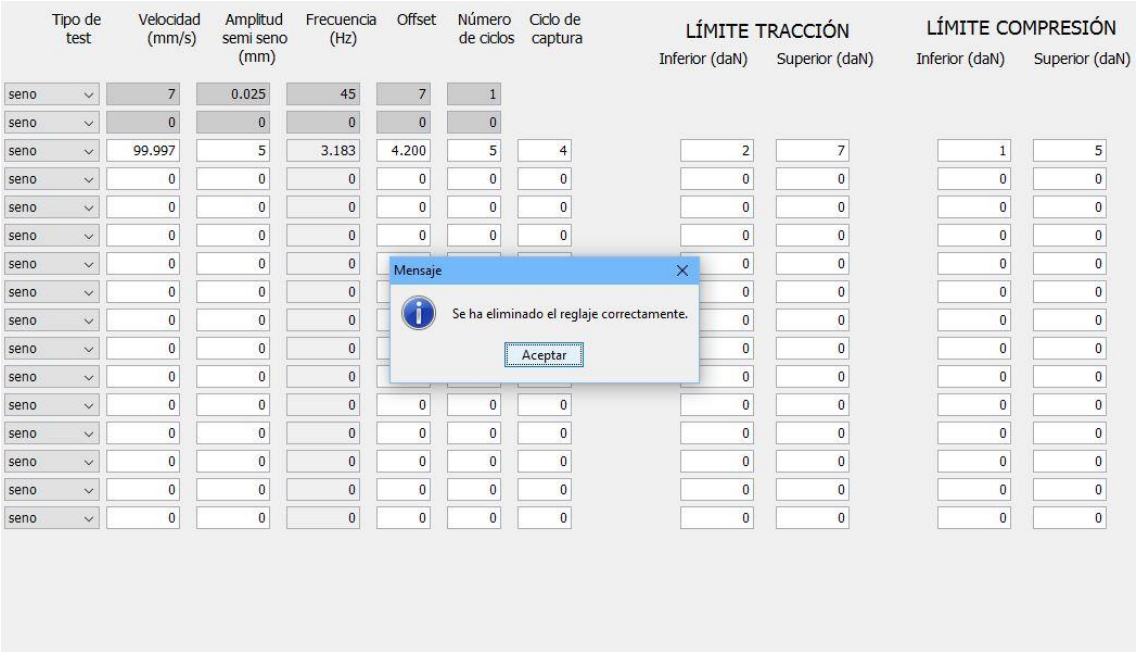


Ilustración 58: Mensaje informativo de reglaje eliminado correctamente

✓ Al pulsar el botón "modificar", suceden las siguientes acciones:

1) Hechas las comprobaciones, se busca el identificador del reglaje mediante el de la equivalencia, gracias al método *getReglajePorEquivalenciaId*() de la clase *Reglaje*.

2) El método del paso 2 devuelve un objeto Reglaje, y de él se obtiene el dato de su versión con el método *getVersion()* y su id con *getId()*.

3) Al igual que en la creación del reglaje, se obtienen todos los datos de la interfaz: cliente, línea, serie, condiciones, precargas…

4) Tampoco se avanza en la modificación si los límites de las condiciones no son coherentes, o si no hay ninguna condición con valores distintos de 0.

5) Si las comprobaciones son correctas, se crea un reglaje. A diferencia de Nuevo reglaje – Detalles, algunos parámetros que eran automáticos, ahora se toman desde la interfaz

(como el campo “revision”, que ya no tiene por qué ser un carácter “-“.) El parámetro “reglaje_id” ya no es un nulo: toma el id del reglaje que se está a punto de modificar.

6) Si la operación se ha completado con éxito, se borra el reglaje anterior con el método *borrar()*, y se informa de ello. Si no, se muestra un mensaje informativo de error.

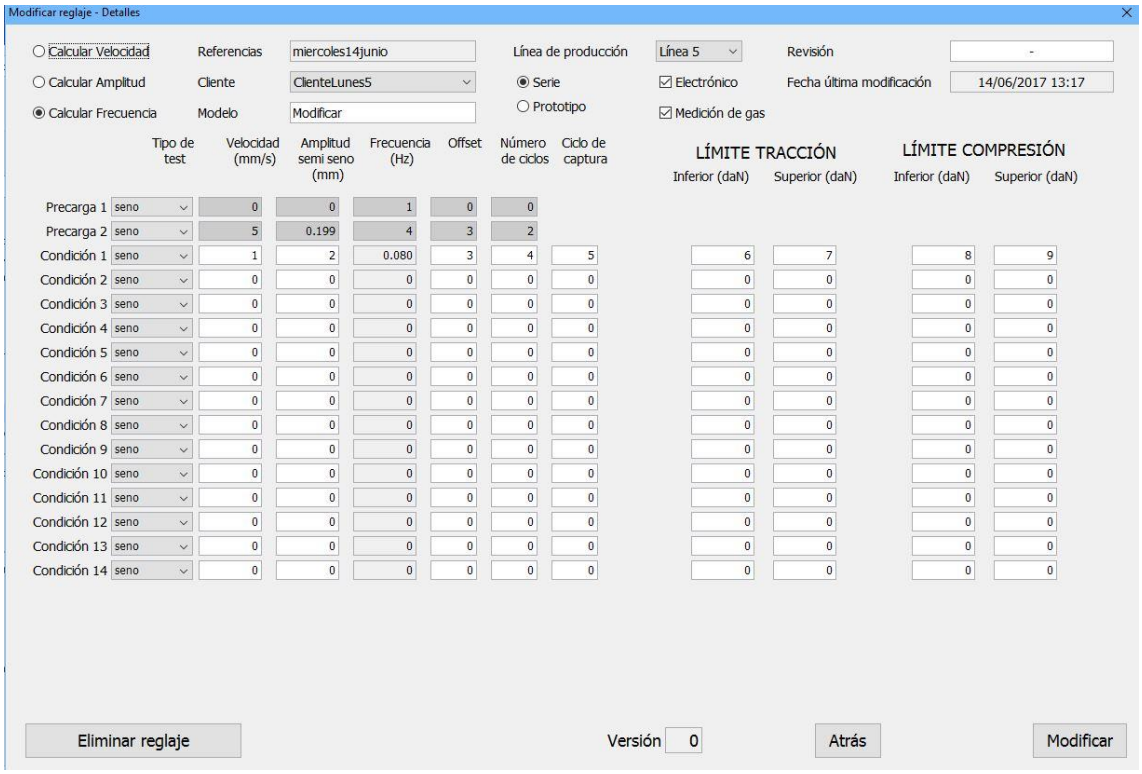


Ilustración 59: un reglaje de ejemplo listo para modificar

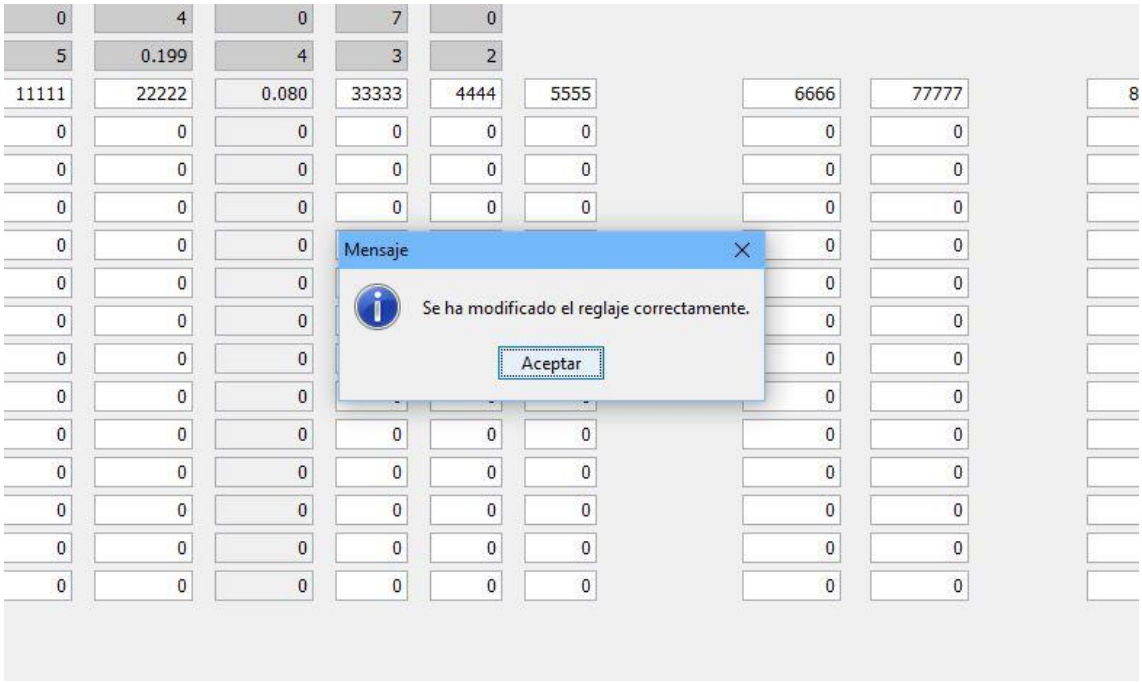


Ilustración 60: mensaje de éxito en la modificación del reglaje

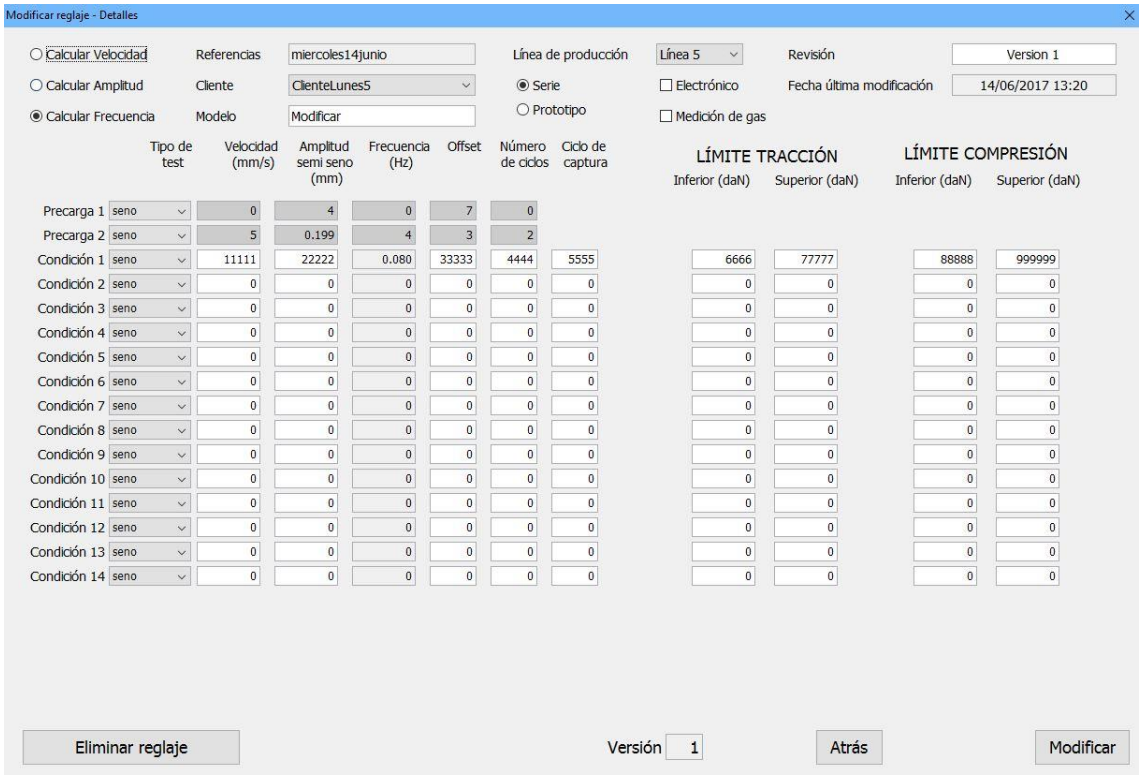


Ilustración 61: reglaje modificado una vez

En las siguientes capturas de SQL Server Management Studio, se aprecian modificaciones hechas sobre un reglaje.

1) En primer lugar, se ha creado la referencia desde Administrar referencias (“equivalencia_id” = 1243).

2) Tras esto, se ha creado un reglaje para ella desde Nuevo reglaje (“id” = 1258).

3) Inmediatamente después, se ha eliminado el reglaje desde Modificar reglaje (su campo se actualiza a “borrado” = 1).

4) Se ha creado otro reglaje desde Nuevo reglaje (“id” = 1259).

5) Se ha modificado este reglaje (“version” = 1, “id” = 1260)

```
select * from reglaje order by id desc
```

00 %

Resultados | Mensajes

	id	equivalencia_id	cliente_id	revision	modelo	electronico	serie	linea_id	fecha	gas	borrado	reglaje_id	version
1	1260	1243	22	Version 1	Modificar	0	1	5	2017-06-14 13:20:21.410	0	0	1259	1
2	1259	1243	22	-	Modificar	1	1	5	2017-06-14 13:17:29.230	1	1	1259	0
3	1258	1243	2	-	Borrar el reglaje	0	1	4	2017-06-14 13:08:29.140	1	1	1258	0
4	1257	611	19	-	nuevoReglaje	0	1	1	2017-06-13 17:25:11.237	1	0	1257	0

Ilustración 62: tabla reglaje

Conforme se hagan más modificaciones sobre el mismo reglaje, se crearán otros donde el número de versión aumentará y “reglaje_id” apuntará siempre al id del anterior.

Resultados | Mensajes

	id	equivalencia_id	cliente_id	revision	modelo	electronico	serie	linea_id	fecha	gas	borrado	reglaje_id	version
1	1262	1243	7	Ahora modifico aquello	Modificar	0	1	15	2017-06-14 13:47:12.927	0	0	1261	3
2	1261	1243	22	Modifico esto	Modificar	0	0	9	2017-06-14 13:46:38.707	0	1	1260	2
3	1260	1243	22	Version 1	Modificar	0	1	5	2017-06-14 13:20:21.410	0	1	1259	1
4	1259	1243	22	-	Modificar	1	1	5	2017-06-14 13:17:29.230	1	1	1259	0
5	1258	1243	2	-	Borrar el reglaje	0	1	4	2017-06-14 13:08:29.140	1	1	1258	0
6	1257	611	19	-	nuevoReglaje	0	1	1	2017-06-13 17:25:11.237	1	0	1257	0

Ilustración 63: reglaje modificado dos veces en la base de datos

Las condiciones y precargas en sus tablas apuntarán al “reglaje_id” al que aplican.

3.8 Menú Administrar referencias

En el software actual de la fábrica se crean referencias a la vez que sus reglajes.

Pero podría ocurrir que haya referencias sin un reglaje asociado. Por ello, se diseña un menú por separado, para gestionar la adición y borrado de referencias.

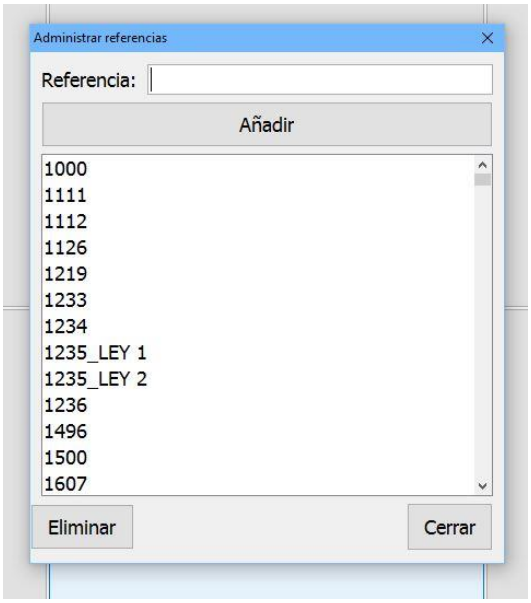


Ilustración 64: Menú Administrar referencias

Antes de mostrar el diálogo, se carga una lista con las referencias no borradas de la tabla *referencia*, gracias al método *getReferencias()* de la clase *Referencia*; en el paquete de entidades.

Las referencias de esta lista se añaden a un elemento JList, que es el elemento semejante a una lista que ocupa casi todo el diálogo.

Se describe el funcionamiento:

- ✓ Se accede a este menú mediante el botón central inferior del Menú principal.
- ✓ El botón de "cerrar" cierra este diálogo mostrando de nuevo el Menú principal.
- ✓ El botón "añadir", hace una serie de comprobaciones antes de añadir una referencia a la base:

1. **La referencia no puede contener el carácter "/"**

Primero, se toma el texto escrito en el campo superior y se eliminan los espacios al inicio o al final del mismo. (Método *trim()*).

Se comprueba si el texto contiene el carácter "/", para evitar que aquí se introduzcan equivalencias. Para eso ya se diseña aparte el menú Administrar equivalencias.

Si lo contiene, muestra un mensaje de error y no hace operaciones. Si no, continúa.

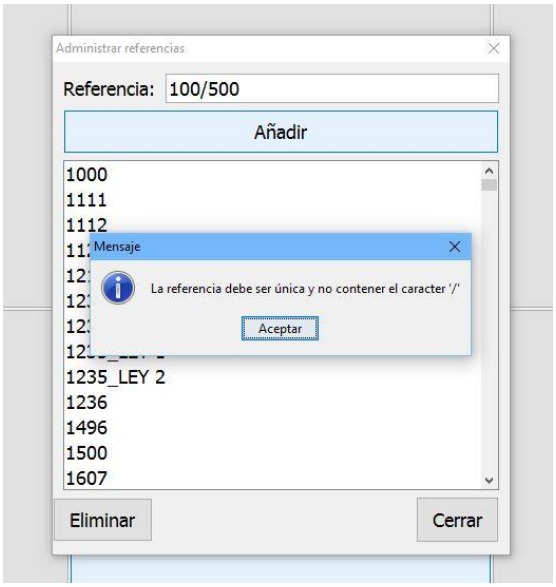


Ilustración 65: Mensaje informativo de error al añadir una referencia con barra.

2. La referencia escrita no puede existir en la base y ser válida

Se comprueba si la cadena de texto está vacía o si ya existe en la base de datos y es válida (campo "borrado" igual a 0). Si alguna de las dos condiciones es cierta, muestra un mensaje de error. En caso negativo, procede a añadir la referencia.

El añadido de la referencia a la base de datos se hace del siguiente modo:

1) Se usa el método *nueva()* de la clase *Referencia*, en el paquete de entidades. Este método crea el registro en la tabla *equivalencia* y en *referencia*.
   Si se insertan correctamente en la base, muestra un mensaje informativo.

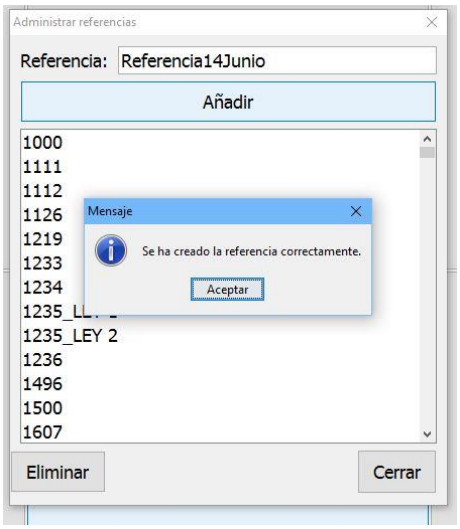
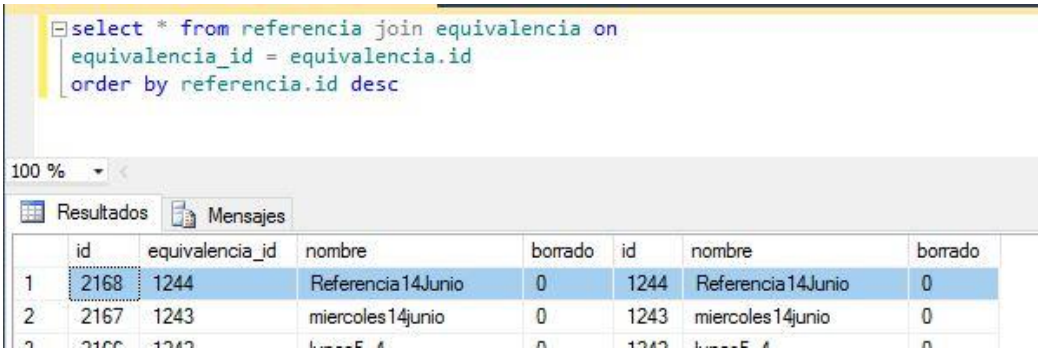


Ilustración 66: mensaje de éxito al crear una referencia

2) Se actualiza la lista volviéndola a cargar con las referencias de la base; entre las que se encuentra la última referencia añadida. Se ha de tener en cuenta, que sólo se ha creado la referencia, así como su equivalencia, pero no un reglaje.



	id	equivalencia_id	nombre	borrado	id	nombre	borrado
1	2168	1244	Referencia14Junio	0	1244	Referencia14Junio	0
2	2167	1243	miercoles14junio	0	1243	miercoles14junio	0

Ilustración 67: unión de tablas equivalencia y referencia, mostrando la nueva referencia

✓ El botón “eliminar” sigue los siguientes pasos:

1) Toma el índice del ítem seleccionado en la lista (*getSelectedIndex()*). Si no se ha seleccionado ninguno, el índice es -1. En ese caso, el botón no hace ninguna acción.

2) Si el índice es mayor o igual a 0, es que hay un ítem seleccionado. Muestra un mensaje de interrogación para confirmar el borrado. Si se clicka en “No”, se cierra el mensaje y no realiza operaciones. Si se clicka en “Sí”, continúa con el borrado.

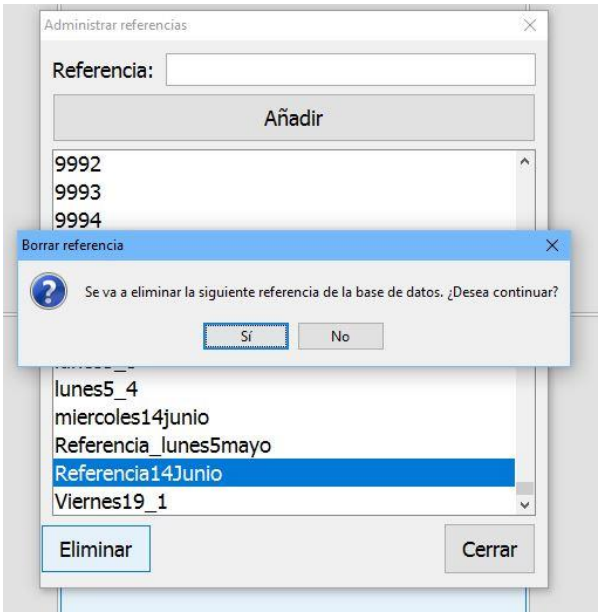


Ilustración 68: mensaje de confirmación antes de borrar una referencia.

3) Antes de borrar una referencia, se comprueba si ésta pertenece a un grupo.

No se permite borrar referencias que forman parte de una equivalencia de grupo puesto que habría incongruencias con la tabla *equivalencia*, y se pide antes desagregarla.

Es decir, si se borrase la referencia 3067, que forma parte de la equivalencia **3067**/3068/3069… no sería adecuado mantener dicha referencia en el nombre.
Esto debería ser gestionado por el apartado de Administrar equivalencias.

Para comprobar si una referencia forma parte de un grupo, se utiliza el método *perteneceAGrupo()* de la clase *Referencia* en el paquete de entidades. Si devuelve un valor verdadero, muestra un mensaje de error y no hace operaciones.

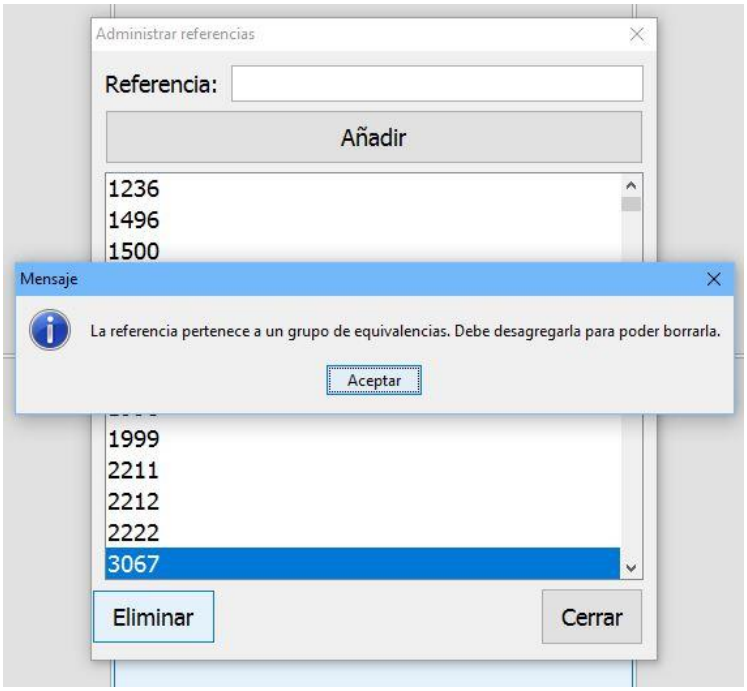


Ilustración 69: mensaje de error al intentar borrar una referencia que forma parte de una equivalencia grupal

4) Si la referencia es única, se puede borrar con el método *borrar()* de la clase *Referencia* en entidades. El método actualiza a valor 1 los campos "borrado" en las tabas *referencia* y *equivalencia*, que corresponden a la referencia seleccionada.

5) Por último, se actualiza la lista, volviéndola a cargar con las referencias válidas de la base.

3.9 Menú Administrar equivalencias

En el software actual de la fábrica se puede añadir un reglaje instaurando hasta cuatro referencias. Ello crearía un grupo de referencias, o equivalencia.

Se diseña un apartado para gestionar estos grupos, pudiendo en los demás trabajar con las referencias (como en los menús Nuevo reglaje y Modificar reglaje).

Con este se menú se puede tanto crear nuevas equivalencias, como modificar las existentes.

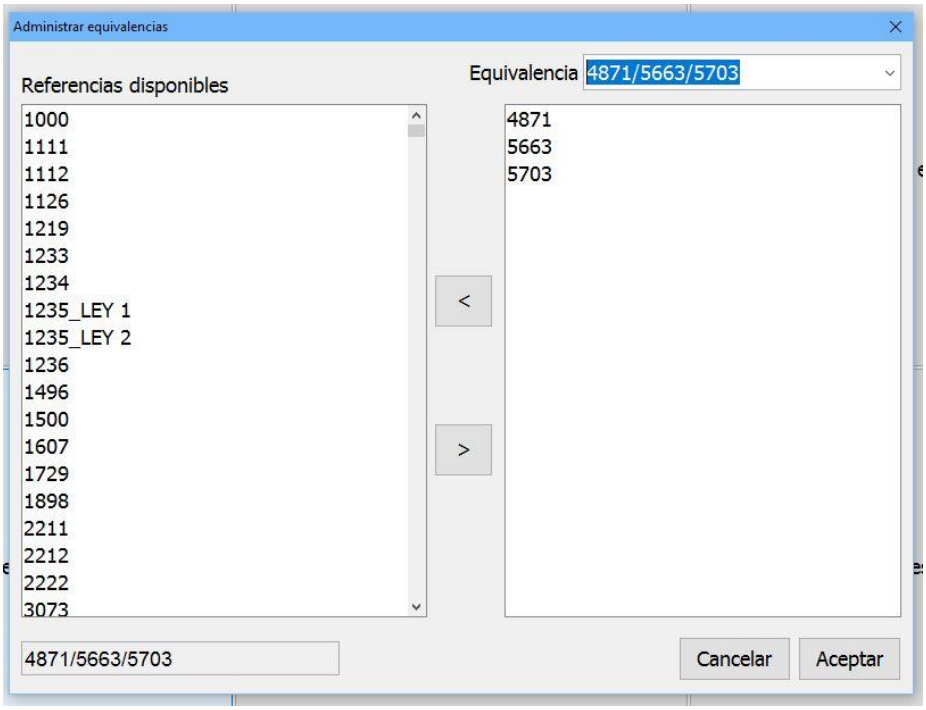


Ilustración 70: Menú Administrar equivalencias

En la lista desplegable de la esquina superior derecha, se muestran todas las equivalencias (sean de una sola referencia o no); con *getEquivalencias().*

En la amplia lista de la izquierda se muestran las equivalencias que son únicas, gracias al método *getReferenciasUnicas()*. Es decir, referencias que no están agrupadas en ninguna equivalencia.

En la lista de la derecha, se muestran cada una de las referencias que hay dentro de la equivalencia seleccionada en el desplegable.

En el recuadro inferior izquierdo se guarda el nombre de la equivalencia tal cual se está modificando, simplemente como información.

El funcionamiento es el siguiente:

- ✓ Se accede aquí desde el botón inferior izquierdo del Menú principal.

- ✓ Al clickar el botón de cancelar, se cierra esta ventana, dejando plenamente visible el menú principal.

- ✓ Hay dos botones con símbolos de flechas.
  El superior, con el símbolo "<" elimina una referencia de la equivalencia.
  El inferior, con el símbolo ">", añade una referencia a la equivalencia.

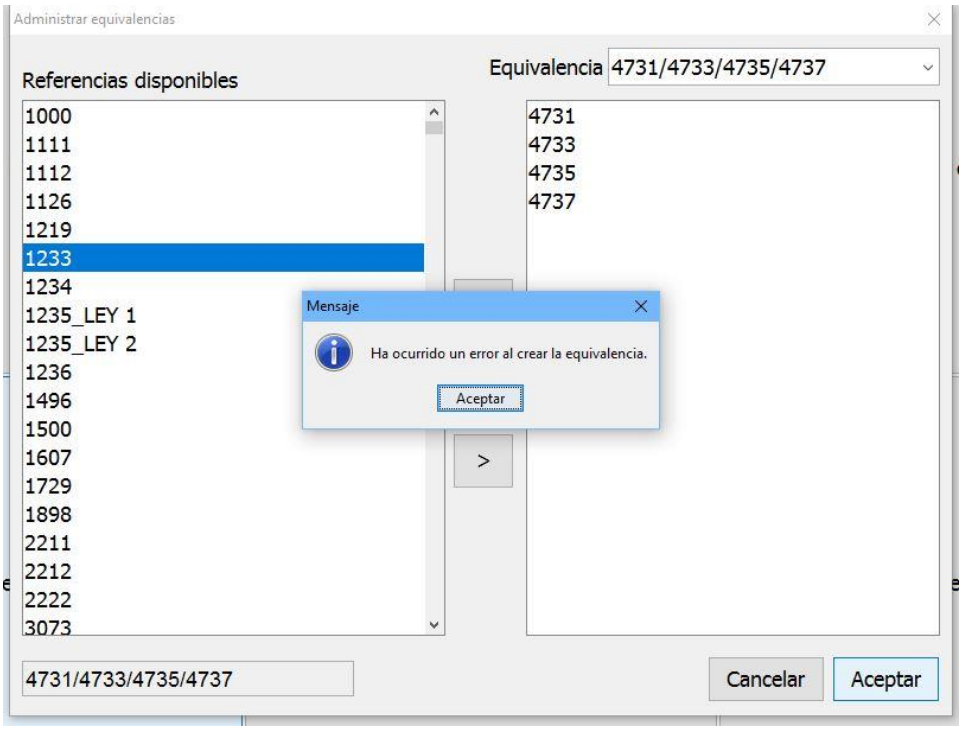


Ilustración 71: mensaje de error en Administrar equivalencias

Analizando el comportamiento de estos botones:

- ✓ El botón de sustraer "<", necesita que esté seleccionada una referencia de la lista derecha o no llevará a cabo ninguna acción. Si lo está, se elimina de la lista derecha y se

añade al final de la lista izquierda. Esto simboliza, que se está aislando una referencia que antes estaba en un grupo.
En la parte inferior, se actualiza el nombre de la nueva equivalencia.

Las dos listas están configuradas desde Netbeans para que no permitan la multiselección (más de una referencia seleccionada); por lo que se añaden/eliminan referencias de una en una.
Tampoco es posible eliminar la última referencia restante de una equivalencia (dejando vacía una equivalencia).
No se trasladan los cambios a la base hasta que se pulse el botón de "aceptar".

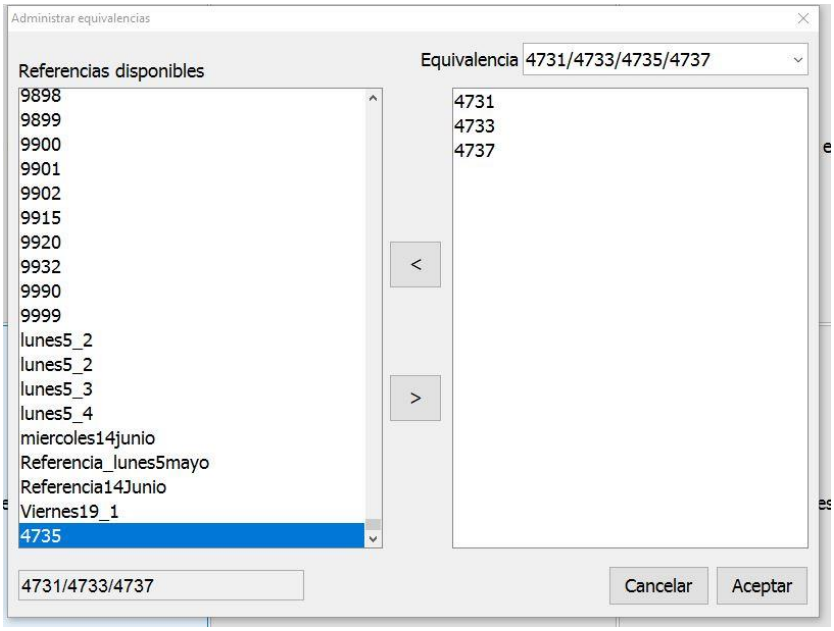


Ilustración 72: equivalencia modificada, se le ha sustraído la referencia 4735

✓ El botón de añadir ">", necesita que esté seleccionada una referencia de la lista izquierda o no llevará a cabo ninguna acción. Si lo está, se elimina de la lista izquierda y

se añade al final de la lista derecha. Esto simboliza, que se está añadiendo una referencia a la equivalencia seleccionada.
En la parte inferior, se actualiza el nombre de la nueva equivalencia.

Se pueden añadir cuantas referencias se quieran.
No se trasladan los cambios a la base hasta que se pulse el botón de "aceptar".

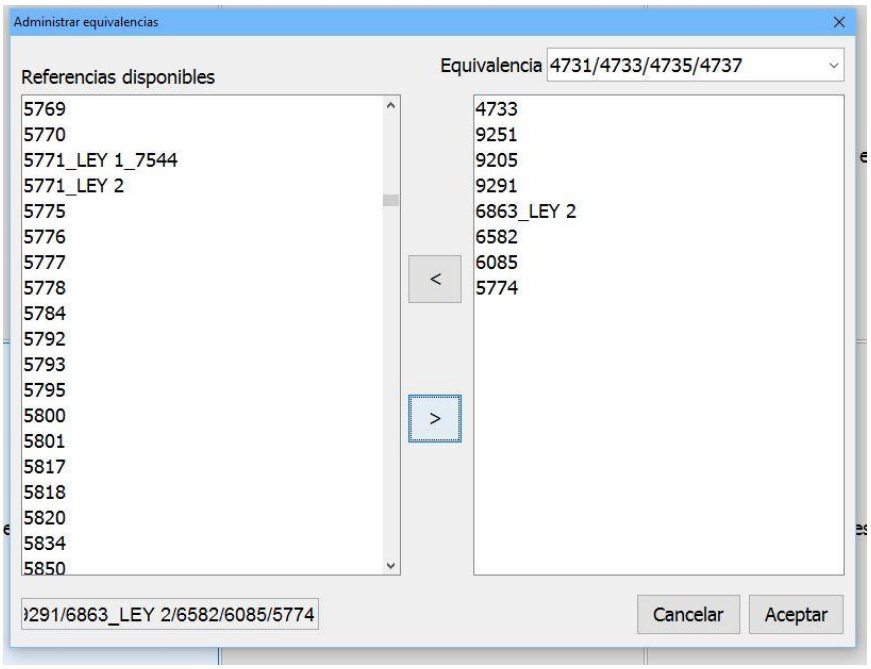


Ilustración 73: equivalencia modificada añadiendo referencias

✓ El botón "aceptar" sigue estos pasos:

1) Se aíslan las referencias que se han eliminado. Es decir, se comprueba qué nuevas referencias han aparecido en la lista izquierda.
   Se recorre la lista de la izquierda y si una referencia pertenece a un grupo (método *perteneceAGrupo()*), se crea una equivalencia única para cada una de ellas (*crearEquivalenciaUnica()*), de la clase *Referencia*.
   Las referencias de la lista son únicas en el inicio por lo que aquella nueva referencia en la lista que pertenezca a un grupo, es la que se ha desagregado.

2) Se toma el nombre de equivalencia de la caja de texto inferior.

3) Se toma una lista de las referencias que están en el cajón derecho y se crea la equivalencia con *crearEquivalencia()*. Este método inserta en *equivalencia* el nombre obtenido en el paso 2.

4) Si las referencias tenían equivalencia única, se actualiza el campo "borrado" de éstas a 1, en la tabla *equivalencia.*

5) Apuntando al id de la equivalencia del paso 3, se actualiza el campo “equivalencia_id” de las referencias agrupadas.

6) Si la operación ha sido exitosa, se informa de ello.

Ejemplo: se tiene la equivalencia 1000/1111/1112. Se quiere eliminar las referencias 1111 y 1112. Y añadir las referencias Ejemplo1, Ejemplo2 y Ejemplo3.

Estado inicial en la interfaz y en la base de datos:

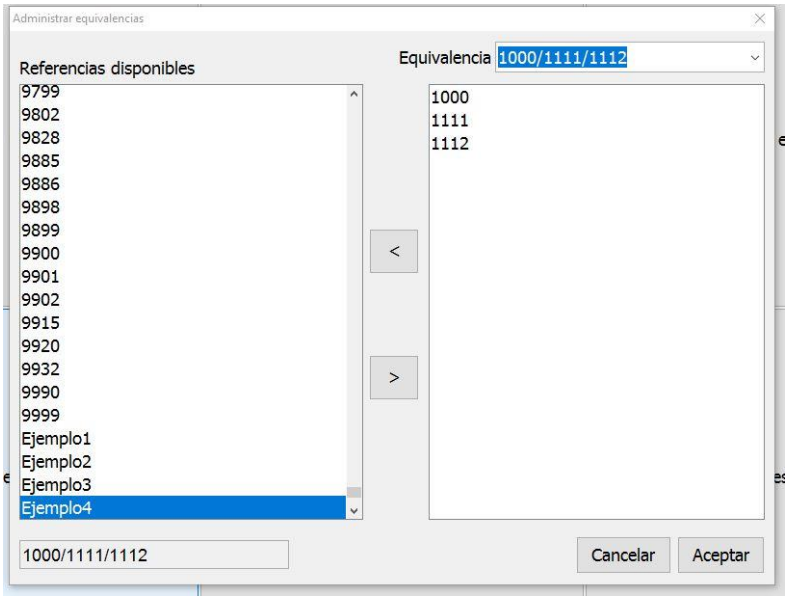
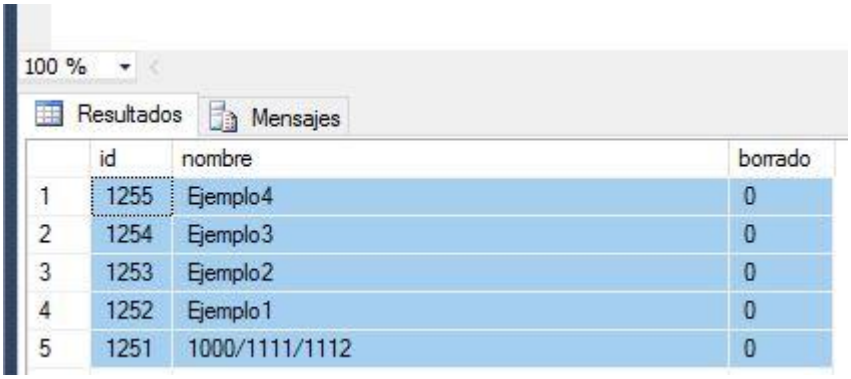


Ilustración 74: ejemplo en el menú Administrar equivalencias



	id	nombre	borrado
1	1255	Ejemplo4	0
2	1254	Ejemplo3	0
3	1253	Ejemplo2	0
4	1252	Ejemplo1	0
5	1251	1000/1111/1112	0

Ilustración 75: tabla equivalencia

Resultados | Mensajes

	id	equivalencia_id	nombre	borrado
1	1	1251	1000	0
2	2	1251	1111	0
3	3	1251	1112	0
4	2169	1252	Ejemplo1	0
5	2170	1253	Ejemplo2	0
6	2171	1254	Ejemplo3	0
7	2172	1255	Ejemplo4	0

Ilustración 76: tabla referencia

Con el cambio hecho desde la interfaz:

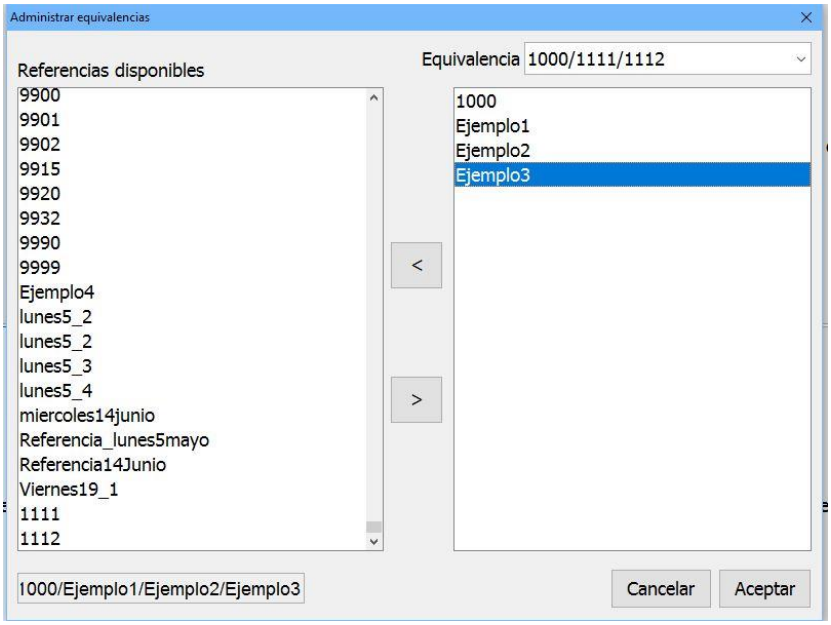


Ilustración 77: antes de ejecutar el cambio

Resultados | Mensajes

	id	nombre	borrado
1	1258	1000/Ejemplo1/Ejemplo2/Ejemplo3	0
2	1257	1112	0
3	1256	1111	0
4	1255	Ejemplo4	0
5	1254	Ejemplo3	1
6	1253	Ejemplo2	1
7	1252	Ejemplo1	1
8	1251	1000/1111/1112	0

Ilustración 78: tabla equivalencia tras la modificación

Resultados | Mensajes

	id	equivalencia_id	nombre	borrado
1	1	1258	1000	0
2	2	1256	1111	0
3	3	1257	1112	0
4	2169	1258	Ejemplo1	0
5	2170	1258	Ejemplo2	0
6	2171	1258	Ejemplo3	0
7	2172	1255	Ejemplo4	0

Ilustración 79: tabla referencia tras la modificación

3.10 Menú Buscar resultados

El menú Buscar resultados sirve para visualizar los resultados numéricos de las pruebas de calidad efectuadas.

Se accede con el botón inferior derecho del Menú principal y muestra en primera instancia un filtro para la búsqueda.

3.10.1 Buscar resultados - Filtro

En el software actual de la fábrica existen dos ventanas distintas para inspeccionar los resultados:

- ✓ Especificando intervalo de fechas

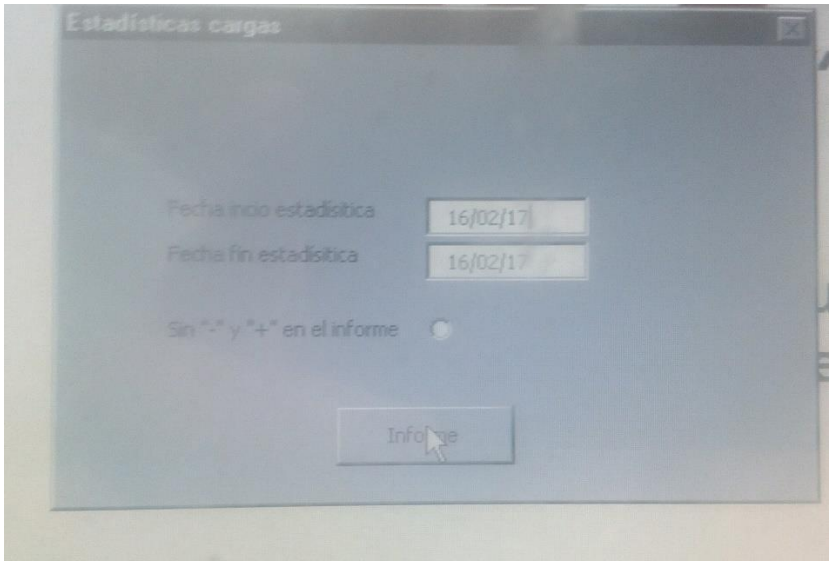


Ilustración 80: Búsqueda de resultados por intervalo de fechas en el software actual

✓ Especificando equivalencia e intervalo de fechas

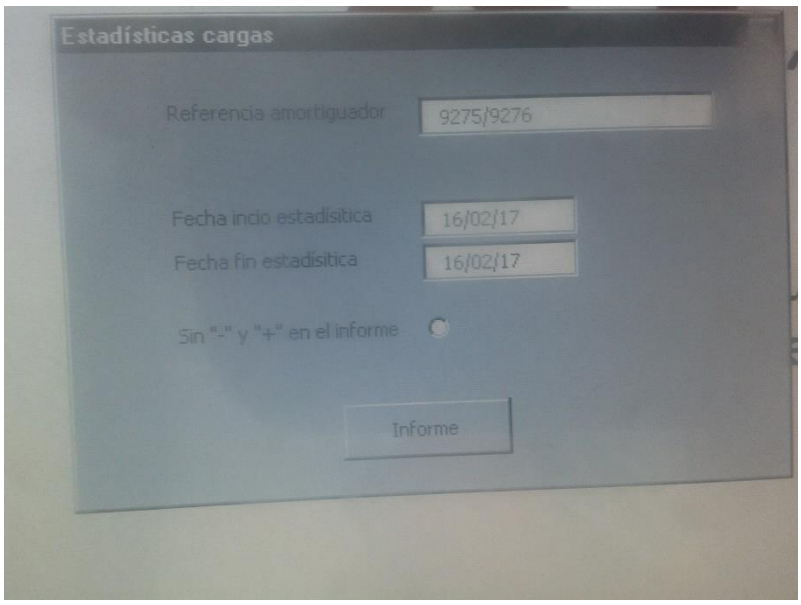


Ilustración 81: Búsqueda de resultados por equivalencia y fechas en el software actual

Se diseña un panel de diálogo semejante al del software actual:

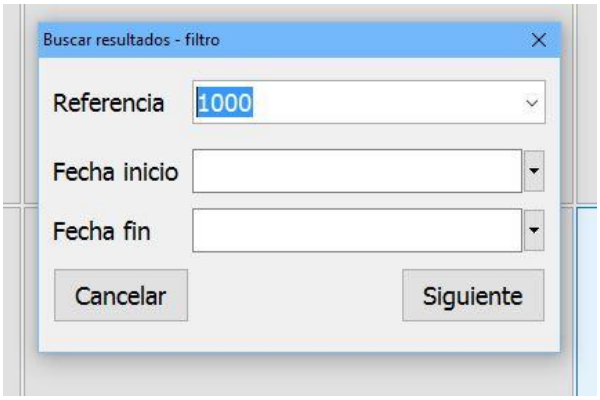


Ilustración 82: Menú Buscar resultados - filtro

Primero se enseña una lista desplegable, y con autocompletado, con los nombres de referencias. Gracias al método del paquete entidad de la clase Referencia: *getReferencias()*.

Fecha de inicio y fecha de fin, sirven para marcar el intervalo bajo el cual se buscarán los resultados. Muestran un calendario que se ha tomado de la librería Swingx, por comodidad para el usuario.

Además, con un calendario se evita la introducción de fechas en diferentes formatos; ya que automáticamente las escribe en formato dd/MM/aa.

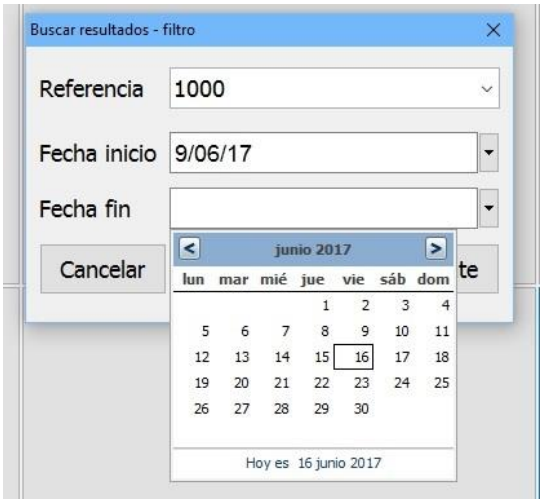


Ilustración 83: calendario en Buscar resultados – Filtro

El funcionamiento del filtro es el siguiente:

- ✓ Se accede aquí desde el botón del Menú principal.
- ✓ Al clickar el botón de “cancelar”, se cierra esta ventana, dejando plenamente visible el menú principal.
- ✓ Al clickar el botón de “siguiente”, se hacen varias comprobaciones:

1. **Los campos de fecha de inicio y fin no deben estar vacíos**

Es obligatorio introducir un intervalo. Si alguno de los dos lo está, muestra un mensaje informativo de error, y no realiza acciones.

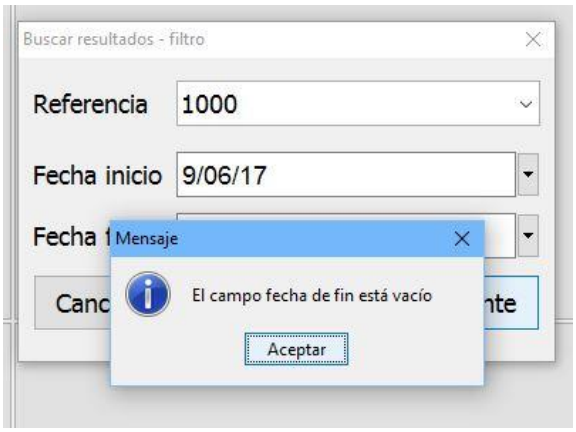


Ilustración 84: mensaje de error informando de que la fecha está vacía en el menú Buscar resultados - Filtro

2. **El campo de fecha de inicio debe ser anterior temporalmente a la fecha de fin.**

Se analiza si el dato de fecha de inicio es mayor (más reciente) que la fecha de fin. Si lo es, se muestra un mensaje de error y no avanza al siguiente diálogo.

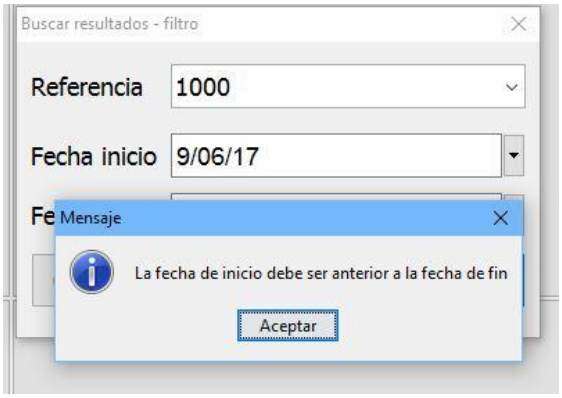


Ilustración 85: fecha de inicio superior a la fecha de fin en el menú Buscar resultados - Filtro

3. Si la caja de texto de la referencia no está vacía, debe ser una referencia válida

Se toma el valor escrito en la caja de texto de la referencia, y se usa el método *getReferencia(nombre)*. Este método busca referencias en la base por su nombre incluso si están borradas. Puede darse el caso de resultados pertenecientes a referencias que se han marcado como borradas.

Si el método devuelve un objeto nulo, la referencia no existe en la base, y se muestra un mensaje de error.

- ✓ Si las comprobaciones son exitosas, se llama al método *buscar()* de la clase *Resultados* en el paquete entidad.
  Si no se encuentran resultados, el método devuelve un booleano falso, en cuyo caso se muestra un mensaje informativo.
  Si se encuentran, se abre el diálogo Resultados.

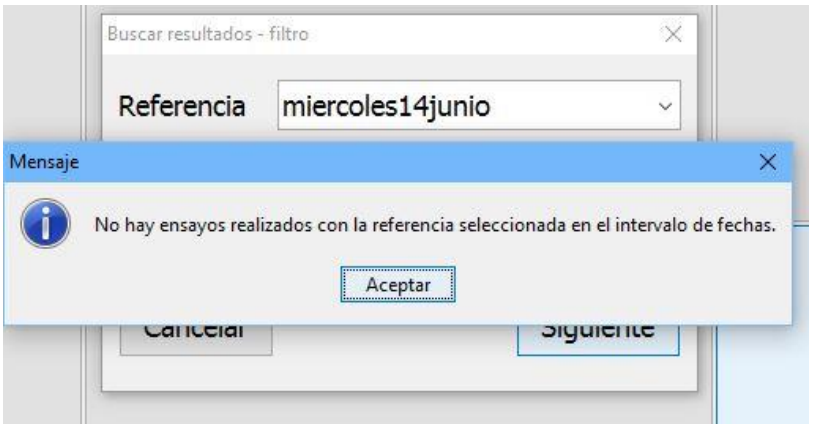
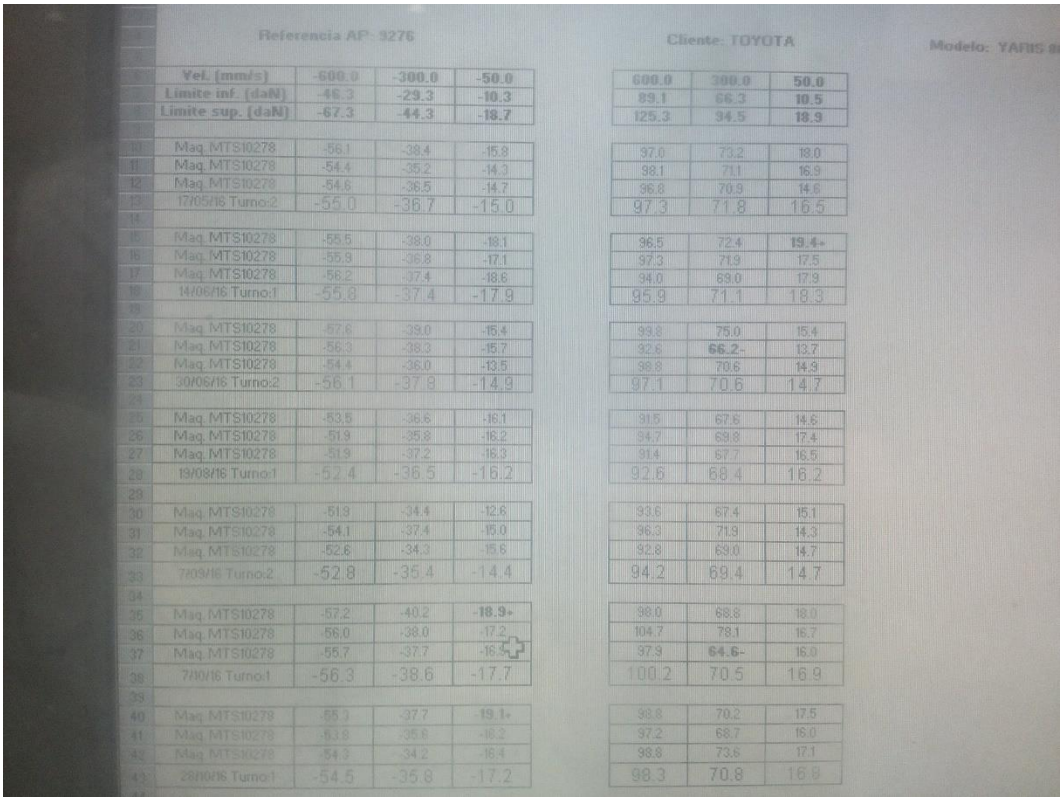


Ilustración 86: mensaje informativo de que no se encuentran resultados con la referencia y fechas especificadas

3.10.2 Resultados

El software actual de la fábrica exporta un archivo con forma de hoja de cálculo, con los resultados buscados.



Referencia AP: 9276 — Cliente: TOYOTA — Modelo: YARIS

Vel. [mm/s]	-600.0	-300.0	-50.0	600.0	300.0	50.0
Límite inf. [daN]	-46.3	-29.3	-10.3	89.1	66.3	10.5
Límite sup. [daN]	-67.3	-44.3	-18.7	125.3	94.5	18.9
Maq. MTS10278	-56.1	-38.4	-15.9	97.0	73.2	18.0
Maq. MTS10278	-54.4	-35.2	-14.3	98.1	71.1	16.9
Maq. MTS10278	-54.6	-36.5	-14.7	96.8	70.9	14.6
17/05/16 Turno:2	-55.0	-36.7	-15.0	97.3	71.8	16.5
Maq. MTS10278	-55.5	-38.0	-19.1	96.5	72.4	19.4-
Maq. MTS10278	-55.9	-36.8	-17.1	97.3	71.9	17.5
Maq. MTS10278	-56.2	-37.4	-18.6	94.0	69.0	17.9
14/06/16 Turno:1	-55.8	-37.4	-17.9	95.9	71.1	18.3
Maq. MTS10278	-57.6	-39.0	-15.4	99.8	75.0	15.4
Maq. MTS10278	-56.3	-38.3	-15.7	92.6	66.2-	13.7
Maq. MTS10278	-54.4	-36.0	-13.5	98.8	70.6	14.9
30/06/16 Turno:2	-56.1	-37.9	-14.9	97.1	70.6	14.7
Maq. MTS10278	-53.5	-36.6	-16.1	91.5	67.6	14.6
Maq. MTS10278	-51.9	-35.8	-16.2	94.7	69.8	17.4
Maq. MTS10278	-51.9	-37.2	-16.3	91.4	67.7	16.5
19/08/16 Turno:1	-52.4	-36.5	-16.2	92.6	68.4	16.2
Maq. MTS10278	-51.9	-34.4	-12.6	93.6	67.4	15.1
Maq. MTS10278	-54.1	-37.4	-15.0	96.3	71.9	14.3
Maq. MTS10278	-52.8	-34.3	-15.6	92.8	69.0	14.7
7/09/16 Turno:2	-52.8	-35.4	-14.4	94.2	69.4	14.7
Maq. MTS10278	-57.2	-40.2	-18.9-	98.0	68.8	18.0
Maq. MTS10278	-56.0	-38.0	-17.2	104.7	78.1	16.7
Maq. MTS10278	-55.7	-37.7	-16.4	97.9	64.6-	16.0
7/10/16 Turno:1	-56.3	-38.6	-17.7	100.2	70.5	16.9
Maq. MTS10278	-55.3	-37.7	-19.1-	98.8	70.2	17.5
Maq. MTS10278	-53.8	-35.6	-16.2	97.2	68.7	16.0
Maq. MTS10278	-54.3	-34.2	-16.4	98.8	73.6	17.1
28/10/16 Turno:1	-54.5	-35.8	-17.2	98.3	70.8	16.8

Ilustración 87: fichero de tablas obtenido con la búsqueda de resultados, en el software actual

En el proyecto se ha decidido mostrar los resultados en la misma interfaz de dos maneras:

- ✓ Especificando un intervalo de fechas

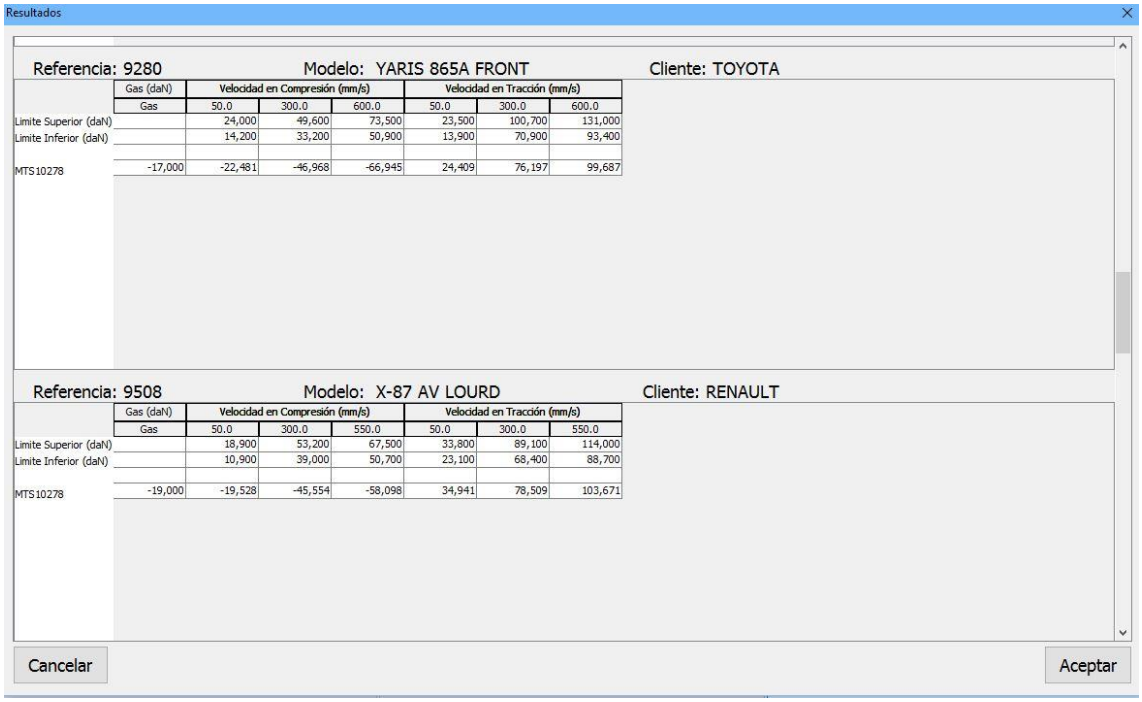


Ilustración 88: Resultados sin especificar referencia en la búsqueda

Se aprecia que para cada ensayo se genera de manera dinámica una cabecera con el nombre de la referencia, el modelo del amortiguador y el cliente; además de cada panel con la tabla de datos.

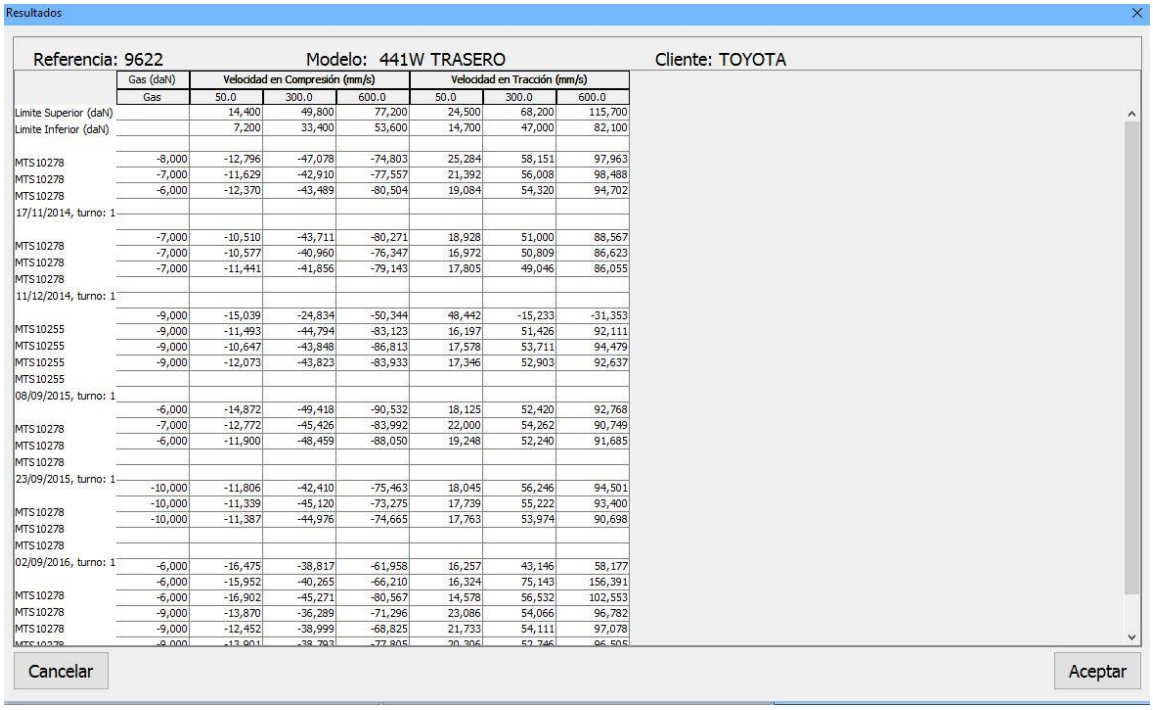
En ella, primero se presentan los datos del reglaje asociado a la referencia. Si tiene gas, aparece una columna con el dato de la fuerza de gas en el ensayo. En caso negativo, no existe la columna.

La tabla se divide en cargas de compresión y tracción, con valores de velocidad en cada una (las condiciones del reglaje). Las celdas contienen los valores límites superior e inferior, dentro de los cuales un amortiguador es considerado correcto (no defectuoso).

Bajo el reglaje, están los resultado. A la izquierda se muestra la máquina en la cual se ha hecho el ensayo, y en las celdas, los valores de fuerza medidos.

La columna de la izquierda, que contiene datos de máquinas, fechas y las etiquetas de límites, es fija. Se debe a que puede haber amortiguadores con muchas condiciones en su reglaje, lo que supone tablas extensas horizontalmente. En esos casos, aparece un scroll (barra deslizadora) horizontal, pero no deja de visualizarse esta columna.

- ✓ Especificando una referencia e intervalo de fechas



	Gas (daN)	Velocidad en Compresión (mm/s)			Velocidad en Tracción (mm/s)		
	Gas	50.0	300.0	600.0	50.0	300.0	600.0
Limite Superior (daN)		14,400	49,800	77,200	24,500	68,200	115,700
Limite Inferior (daN)		7,200	33,400	53,600	14,700	47,000	82,100
MTS10278	-8,000	-12,796	-47,078	-74,803	25,284	58,151	97,963
MTS10278	-7,000	-11,629	-42,910	-77,557	21,392	56,008	98,488
MTS10278	-6,000	-12,370	-43,489	-80,504	19,084	54,320	94,702
17/11/2014, turno: 1							
	-7,000	-10,510	-43,711	-80,271	18,928	51,000	88,567
MTS10278	-7,000	-10,577	-40,960	-76,347	16,972	50,809	86,623
MTS10278	-7,000	-11,441	-41,856	-79,143	17,805	49,046	86,055
MTS10278							
11/12/2014, turno: 1							
	-9,000	-15,039	-24,834	-50,344	48,442	-15,233	-31,353
MTS10255	-9,000	-11,493	-44,794	-83,123	16,197	51,426	92,111
MTS10255	-9,000	-10,647	-43,848	-86,813	17,578	53,711	94,479
MTS10255	-9,000	-12,073	-43,823	-83,933	17,346	52,903	92,637
MTS10255							
08/09/2015, turno: 1							
	-6,000	-14,872	-49,418	-90,532	18,125	52,420	92,768
MTS10278	-7,000	-12,772	-45,426	-83,992	22,000	54,262	90,749
MTS10278	-6,000	-11,900	-48,459	-88,050	19,248	52,240	91,685
MTS10278							
23/09/2015, turno: 1	-10,000	-11,806	-42,410	-75,463	18,045	56,246	94,501
	-10,000	-11,339	-45,120	-73,275	17,739	55,222	93,400
MTS10278	-10,000	-11,387	-44,976	-74,665	17,763	53,974	90,698
MTS10278							
MTS10278							
02/09/2016, turno: 1	-6,000	-16,475	-38,817	-61,958	16,257	43,146	58,177
	-6,000	-15,952	-40,265	-66,210	16,324	75,143	156,391
MTS10278	-6,000	-16,902	-45,271	-80,567	14,578	56,532	102,553
MTS10278	-9,000	-13,870	-36,289	-71,296	23,086	54,066	96,782
MTS10278	-9,000	-12,452	-38,999	-68,825	21,733	54,111	97,078
MTS10278	-8,000	-13,901	-38,793	-77,805	20,306	52,746	96,505

Ilustración 89: Resultados especificando referencia en la búsqueda

En el caso de que la referencia esté especificada en el filtro, se construye solamente una gran tabla con los datos.

Se agrupan los ensayos por fecha y turno, intercalando una fila vacía cuando una fecha y turno sean distintos del ensayo anterior.

Las tablas se han realizado gracias a unas clases programadas por el coordinador técnico Iñigo Sola. Se han reacomodado de un proyecto anterior para el mismo cliente; que requería también una interfaz y presentación de resultados y estadísticas.

Estas tablas sólo necesitan especificar los datos de las cabeceras verticales y horizontales; y los datos de celdas.

Los datos se obtienen gracias a los métodos de la clase entidad *Ensayo*. De ellos, se extraen las cargas, que se guardan en una matriz de dos dimensiones de decimales (double[][]).

Desde este panel de diálogo hay dos opciones:

- ✓ Presionando el botón “cancelar” se cierra este diálogo y se visibiliza el menú Buscar resultados – filtro; con las últimas condiciones de fecha y referencia introducidas.

- ✓ Presionando el botón “aceptar”, se cierra este diálogo y se visualiza el menú principal.

3.11 Menú Lanzar ensayo

Desde el menú principal, el botón derecho superior accede al inicio de un ensayo.

Desde este menú se selecciona la referencia del amortiguador sobre el que se hará un test de calidad.

3.11.1 Lanzar ensayo – Seleccionar referencia

Al pulsar el botón que dirige al lanzamiento de un ensayo, se muestra primero un selector de referencia.

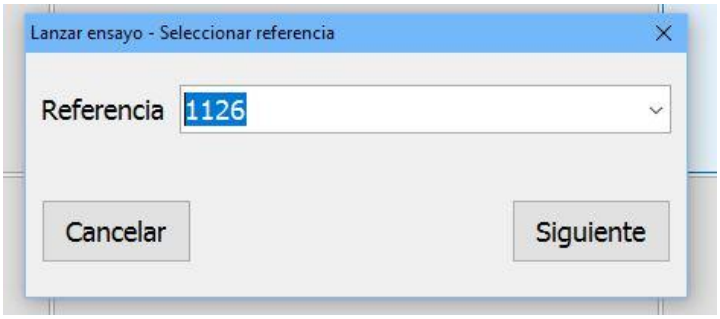


Ilustración 90: menú Lanzar ensayo - Seleccionar referencia

Se muestran en la lista los nombres de referencias válidas y con reglaje asociado.

El funcionamiento es idéntico al menú Modificar reglaje – Seleccionar referencia.

Lo único que lo distingue, es su título y el parámetro que envía al constructor del siguiente diálogo. En esta situación, el booleano de lanzar ensayo será “verdadero”.

3.11.2 Lanzar ensayo – Detalles

Antes de lanzar un ensayo, se presentan los detalles del reglaje para el amortiguador seleccionado.

Ilustración 91: fotografía del software actual de la fábrica con el reglaje antes de lanzar un ensayo

Lanzar ensayo - Detalles

Referencias	1126		Línea de producción	Línea 2		Revisión	-					
Cliente	RENAULT		Serie	Electrónico		Fecha última modificación	21/05/2015 00:00					
Modelo	X-52 DEL.		Prototipo	Medición de gas								
	Tipo de test	Velocidad (mm/s)	Amplitud semi seno (mm)	Frecuencia (Hz)	Offset	Número de ciclos	Ciclo de captura	LÍMITE TRACCIÓN Inferior (daN)	LÍMITE TRACCIÓN Superior (daN)	LÍMITE COMPRESIÓN Inferior (daN)	LÍMITE COMPRESIÓN Superior (daN)	
Precarga 1	seno	550	30	2.920	10	0						
Precarga 2	seno	0	0	0	0	0						
Condición 1	seno	300	25	1.910	0	3	2	60	120	30	50	
Condición 2	seno	550	25	3.500	0	3	2	90	150	40	65	
Condición 3	seno	600	25	3.820	0	3	2	10	200	10	100	
Condición 4	seno	0	0	0	0	0	0	0	0	0	0	
Condición 5	seno	0	0	0	0	0	0	0	0	0	0	
Condición 6	seno	0	0	0	0	0	0	0	0	0	0	
Condición 7	seno	0	0	0	0	0	0	0	0	0	0	
Condición 8	seno	0	0	0	0	0	0	0	0	0	0	
Condición 9	seno	0	0	0	0	0	0	0	0	0	0	
Condición 10	seno	0	0	0	0	0	0	0	0	0	0	
Condición 11	seno	0	0	0	0	0	0	0	0	0	0	
Condición 12	seno	0	0	0	0	0	0	0	0	0	0	
Condición 13	seno	0	0	0	0	0	0	0	0	0	0	
Condición 14	seno	0	0	0	0	0	0	0	0	0	0	

Ilustración 92: Menú Lanzar ensayo - Detalles

El funcionamiento del menú es idéntico al de Modificar reglaje – Detalles.

Al iniciarse, se le ha transmitido al constructor que se trata de un ensayo; por lo que cambiará el título y la diferencia fundamental con el menú Modificar reglaje: que ningún elemento es editable. La presentación de los datos de reglaje tiene función informativa.

Para bloquear la edición de los elementos se llama a unos métodos creados especialmente para este fin, cuya única tarea es inhabilitarlos: *setEnabled* ó *setEditable (false)*, dependiendo del componente.

El funcionamiento de este menú es el siguiente:

- ✓ Se accede aquí desde la selección de referencia en Lanzar ensayo – Seleccionar referencia

- ✓ Al pulsar el botón "cancelar", se cierra la ventana Lanzar ensayo – Detalles y se activa la visibilidad de la anterior: Lanzar ensayo – Seleccionar referencia.

- ✓ Al pulsar el botón "aceptar" debería dar comienzo el ensayo.

Esta acción no se ha podido completar por falta de tiempo en el proyecto.

Sin embargo, el planteamiento sería:

1) Enviar al dispositivo controlador las directrices de las distintas condiciones del reglaje. Es decir, las velocidades, amplitudes, etc. Con las que se ha de configurar la máquina ejecutora de ensayos de calidad.

2) Tras recibir los datos de la máquina; se analizarían los valores de fuerza en compresión y tracción medidos, para las distintas condiciones. Comparándolos con los establecidos en su correspondiente tabla *reglaje*, se determinaría si el amortiguador es defectuoso o no.

3) Debería crearse un registro en la tabla *ensayo* con el dato de carga de gas imprimido, fecha del ensayo, máquina en la que se ha realizado, etc. Y la calificación de defectuoso/no defectuoso.

4) Ya creado el registro en la tabla de *ensayo,* deberían almacenarse las filas en la tabla *resultado* con los datos de fuerzas en las distintas condiciones.

4. Adquisición de datos

A causa de las incidencias sucedidas a lo largo del diseño y creación de la base de datos nueva, junto con la migración de datos y el funcionamiento pleno de la interfaz gráfica; no hubo tiempo suficiente para desarrollar este apartado final.

Otro añadido es que, el dispositivo elegido de National Instruments no era programable en lenguaje Java. Sí en lenguajes como ANSI C, C# .NET, VB .NET…

Se pueden comprobar estas características en su página de producto [13].

A pesar de ello, es posible programar una pasarela que funcione como traductor de lenguajes.

Dadas estas complicaciones, se ha dejado planteado escribir unas clases en Java para la simulación de comunicación entre la interfaz y el dispositivo.

Consisten en el envío de información desde la interfaz, al confirmar el lanzamiento de un ensayo desde el menú Lanzar ensayo – Detalles. Y la posterior recogida de unos datos, por ahora sin una función real, que deben almacenarse correctamente en las tablas *ensayo* y *resultado*.

5. Líneas futuras

Este apartado tiene como fin proponer algunas ideas para continuar y enriquecer el proyecto.

Algunas de ellas son:

1. **Cumplir todos los objetivos del proyecto**

Uno de los objetivos era poder enviar a la máquina de ensayos las órdenes para ejecutar los tests de calidad; a través de un dispositivo hardware. Tras ello, recoger los resultados y almacenarlos en la base de datos.

Debido a la extensión temporal de los anteriores puntos, no dio tiempo a completar el último.

Para ello será necesario construir las pasarelas para programar el control del dispositivo en un lenguaje disponible y comprobar su correcto funcionamiento mecánico, una vez instalado el sistema completo en la fábrica.

2. **Acelerar el proceso de migración de datos de la base original a la base nueva, en la fábrica**

Para migrar los datos de la fábrica, se deben seguir los siguientes pasos:

1) Editar el archivo de configuración, para indicar la localización y nombre de las bases de datos en la máquina.

2) Ejecutar el script de creación de la base de datos nueva. Para ello, es necesario haberlo generado antes en la máquina donde se está llevando a cabo el desarrollo del sistema.

3) Ejecutar el proyecto Java desde la consola de la máquina, anteponiendo el parámetro –*migracion.*

Este proceso resulta algo costoso y rudimentario. Lo ideal sería instalar el sistema en la máquina cliente en uno o dos pasos como máximo.

Para hacerlo, una solución sería incluir en las clases Java la ejecución del script que crea la base nueva.

Otro inconveniente, más serio, es el tiempo empleado en la migración.

Tal y como está programada, es muy costosa temporalmente: tarda de 5 a 7 minutos a causa de sus numerosos registros (en especial con la tabla *resultado*); bucles que recorren listas y arrays, concatenaciones, y comprobaciones.

Muchos de los datos de las filas (pero no toda la fila), se sabe que son números iguales a 0. Con un conocimiento total de los datos, se podrían ignorar los registros adecuados sin repercusión para el cliente. Disminuyendo el tamaño del archivo y el tiempo de migración.

También, puesto que la migración se programó al inicio del proyecto, las clases mezclan conexiones a bases de datos con otra lógica. Sería más ordenado separar las clases en paquetes, al igual que con la interfaz. Por un lado, tener unas clases de entidad que representen a las tablas de la base de datos original, con métodos que se encarguen de las consultas. (Selección, modificación, borrado, etc.).

Y por otro, la lógica de la migración: separado de equivalencias en referencias, ordenado de velocidades del reglaje, etc.

3. Optimizar la búsqueda de resultados

Al contar con tantos registros en la tabla *resultado*, el tiempo de presentación de los mismos es largo, y la búsqueda ineficiente.

La consulta en la base recorre todos los registros y selecciona sólo aquellos que cumplen las restricciones pedidas (fecha y tal vez referencia). Si los resultados estuviesen ordenados (en este caso, por ejemplo, por fecha), sería más simple descartar en una primera selección todos los registros fuera del intervalo de fechas. Es el objetivo de la indexación y las búsquedas dicotómicas.

La indexación consiste en dar una estructura a los datos que provee un acceso rápido y único a los registros; y puede ser una función sobre un campo. En este caso se usaría sobre las fechas [14].

Con indexación se pueden realizar búsquedas dicotómicas, que consisten en dividir en mitades los registros ya ordenados, y descargar una opción en cada iteración hasta encontrar el resultado deseado. De este modo, se reducen exponencialmente las iteraciones necesarias.

4. **Perfeccionar la interfaz gráfica**

El apartado visual y de manejo de errores de la interfaz es bastante simple.

- ✓ Una mejora podría consistir en redirigir al menú correspondiente cuando se cometan algunos fallos. Por ejemplo, redirigir al menú Modificar reglaje cuando se introduce una referencia con reglaje en el menú Nuevo reglaje.

- ✓ Las listas desplegables que se autocompletan no son del todo cómodas cuando ya ha encontrado una coincidencia y se intenta borrar el texto para rectificar. Es necesario introducir un carácter que no encuentre, y después borrar. Se mejoraría utilizando un elemento de alguna otra librería específica o mejorando con código del que ya se tiene.

- ✓ Sería muy interesante que, en el mostrado de resultados, se añadiesen gráficos, estadísticas, y exportación a archivos. Como pdf, Excel…

- ✓ Se podría embellecer la interfaz con una estética propia, en lugar de las que existen por defecto, añadir los logos de la empresa (cliente y/o desarrolladora), animaciones cortas y suaves en la transición de ventanas.

- ✓ Como se puede instalar el sistema en distintos ordenadores o monitores de la fábrica, los iconos podrían ser de formato de dibujos vectoriales; evitando el pixelado por baja resolución de pantalla.

- ✓ Al crear distintas tablas en la búsqueda por fechas, el scroll vertical no es del todo cómodo (no tiene funcionalidad una vez el ratón está en el área de la tabla). Además, hay un pequeño desfase entre las filas de las celdas, y de la columna fija en la izquierda. En sistemas Linux se visualiza correctamente, pero no en Windows. Es necesario reparar estos detalles visuales.

6. Conclusiones

En definitiva, este proyecto ha presentado un marco de trabajo muy específico, que es el del desarrollo software en entorno industrial; distinto al que un estudiante de ingeniería de telecomunicación puede estar acostumbrado.

Es evidente la necesidad de claridad y orden en el proceso, así como las comprobaciones cada vez que se termina una tarea.

La temática ha sido muy variada, incluyendo conceptos de desarrollo software, de información (bases de datos), conceptos de mecánica, … incluso de relaciones laborales con el cliente.

Al ser un proyecto real, conlleva una planificación, una consecución de objetivos, un seguimiento y unos imprevistos.

Ha sido una experiencia muy interesante y a pesar de que el proyecto no ha llegado a finalizarse, se han alcanzado las metas esperadas en su comienzo.

Sobre todo, la conclusión es que ha aportado gran cantidad de conocimiento a la estudiante.

Bibliografía

[1] "Diferencias entre los formatos de archivo ACCDB y MDB", Soporte de Microsoft Access, *Microsoft*, 2017. [En línea]. Disponible en: https://goo.gl/dVrdhz [Accedido: 25-abr-2017]

[2] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, 1983.

[3] "Normalización de bases de datos", *Wikipedia,* 2017. [En línea]. Disponible en: https://goo.gl/P6nQdY [Accedido: 7-jun-2017]

[4] "Capitalization styles", Microsoft .NET Documentation, *Microsoft ,* 2017. [En línea]. Disponible en: https://goo.gl/6SGkuy [Accedido: 26-abr-2017]

[5] "Data types", Microsoft Docs SQL, *Microsoft,* 2017. [En línea]. Disponible en: https://goo.gl/rTlHVb [Accedido: 26-abr-2017]

[6] "SQL Server Management Studio", Herramientas de SQL Server, *Microsoft*, 2017. [En línea]. Disponible en: https://goo.gl/14bfHH [Accedido: 8-may-2017]

[7] "Microsoft SQL Server Migration Assistant for Access", Microsoft Downloads, *Microsoft,* 2017. [En línea]. Disponible en: https://goo.gl/x3vrOg [Accedido: 8-may-2017]

[8] "Microsoft JDBC Driver for SQL Server", Microsoft Docs, *Microsoft*, 2017. [En línea]. Disponible en: https://goo.gl/yZGNjJ [Accedido: 22-may-2017]

[9] "Using Prepared Statements", The Java Tutorials, Java Doc, *Oracle,* 2015. [En línea]. Disponible en: https://goo.gl/7UAex [Accedido: 14-jun-2017]

[10] "Data Access Object", *Wikipedia*, 2017. [En línea]. Disponible en: https://goo.gl/tfjrQA [Accedido: 10-jun-2017]

[11] "Model – View – Controller", *Wikipedia*, 2017. [En línea]. Disponible en: https://goo.gl/R9WCw [Accedido: 10-jun-2017]

[12] Iconos de libre uso, *varias fuentes*. [En línea]. Disponible en: http://www.freeiconspng.com , http://simpleicon.com, http://www.freepik.es [Accedido: 14-jun-2017]

[13] “Dispositivo DAQ multifunción USB-6002”, Productos y Servicios, *National Instruments Corporation*, 2017. [En línea] Disponible en: https://goo.gl/RTXW25 [Accedido: 17-jun-2017]

[14] “Índice (bases de datos)”, *Wikipedia*, 2017. [En línea] Disponible en: https://goo.gl/NzNNrz [Accedido: 17-jun-2017]

[15] “Reglar”, de reglaje, *Real Academia Española (RAE),*2017. [En línea]. Disponible en: http://dle.rae.es/?id=VjDIP4t [Accedido: 5-mayo-2017]

[16] “Ajusta la suspensión a tu peso”, Bici Fácil, *Motorpress Ibérica*, 2017. [En línea]. Disponible en: https://goo.gl/fQGx82 [Accedido: 17-jun-2017]

[17] “Comportamiento básico de la motocicleta: reglaje de suspensiones”, *Circula Seguro*, 2017. [En línea]. Disponible en: https://goo.gl/6jFE7t [Accedido: 17-jun-2017]

[18] “Common questions”, Technical Information, *Kyb Europe G.M.B.H.,* 2008. [En línea] Disponible en: https://goo.gl/Pk4mRp [Accedido: 15-jun-2017]

[19] “Wekan”, *GitHub Inc.*, 2017.[En línea]. Disponible en: https://wekan.github.io/ [Accedido: 30-mayo-2017]

[20] “Metodología Kanban”, *Wikipedia*, 2017.[En línea] . Disponible en: https://es.wikipedia.org/wiki/Kanban [Accedido: 24-mayo-2017]

[21] “Kanban Tool”, *Shore Labs*, 2009-2017.[En línea]. Disponible en: http://kanbantool.com/es/metodologia-kanban [Accedido: 30-mayo-2017]

[22] “What is version control?”, Atlassian Git Tutorials, *Atlassian*, 2017. [En línea]. Disponible en: https://goo.gl/n87P36 [Accedido: 30-mayo-2017]

[23] “What is Git?”, Atlassian Git Tutorials, *Atlassian*, 2017. [En línea]. Disponible en: https://goo.gl/DxaED8 [Accedido: 30-mayo-2017]

[24] “Simple words for a Gitlab newbie” , *Gitlab Inc.*, 2017. [En línea]. Disponible en: https://goo.gl/a2EOY4 [Accedido: 30-mayo-2017]

[25] “SourceTree”, *Atlassian* ,2017. [En línea]. Disponible en: https://www.sourcetreeapp.com/ [Accedido: 30-mayo-2017]

[26] “gitignore.io”, *Joe Blau*, 2017. [En línea] Disponible en: https://www.gitignore.io/ [Accedido: 17-jun-2017]

[27] “¿Qué es Gradle?”, Arquitectura Java, *Cecilio Álvarez Caules*, 2015. [En línea] Disponible en: http://www.arquitecturajava.com/que-es-gradle/ [Accedido: 15-jun-2017]

[28] “Apache Maven Project”, *The Apache Software Foundation*, 2017. [En línea]. Disponible en: https://maven.apache.org/ [Accedido: 17-jun-2017]

[29] “Artifactory Open Source”, *JFrog Open Source Solutions,*2017. [En línea]. Disponibe en: https://www.jfrog.com/open-source/ [Accedido: 17-jun-2017]

[30] “Máquinas virtuales”, *Wikipedia,* 2017. [En línea]. Disponible en: https://goo.gl/Tq4uE4 [Accedido: 16-abr-2017]

[31] “VirtualBox”, *Oracle,* 2017. [En línea]. Disponible en: https://www.virtualbox.org/ [Accedido: 16-abr-2017]

[32] “Guest Additions”, Manual de VirtualBox , capítulo 4, *Oracle*. [En línea]. Disponible en: https://www.virtualbox.org/manual/ch04.html [Accedido: 16-abr-2017]

[33] “DIA Diagram Editor”, *The DIA Developers*, 2014. [En línea]. Disponible en: http://dia-installer.de/ [Accedido: 17-jun-2017]

[34] “SQL”, *Wikipedia*, 2017. [En línea]. Disponible en: https://es.wikipedia.org/wiki/SQL [Accedido: 18-jun-2017]

[35] “SQL Tutorial”, *W3Schools*, 2017. [En línea]. Disponible en: https://www.w3schools.com/sql/default.asp [Accedido: 14-jun-2017]

[36] “Create view”, Microsoft Docs, *Microsoft*, 2017. [En línea] Disponible en: https://goo.gl/gMFHBH [Accedido: 18-jun-2017]

[37] “Aggregate Functions”, Microsoft Docs, *Microsoft*, 2017. [En línea] Disponible en: https://goo.gl/Gj2z6e [Accedido: 18-jun-2017]

[38] “Teoría de conjuntos”, *Wikipedia*, 2017. [En línea]. Disponible en: https://goo.gl/MmxCqV [Accedido: 18-jun-2017]

[39] “Create Trigger”, Microsoft Docs, *Microsoft*, 2017. [En línea] Disponible en: https://goo.gl/bApvqP [Accedido: 18-jun-2017]

[40] “Generar un script (SQL Server Management Studio)”, Microsoft TechNet, *Microsoft*, 2017. [En línea] Disponible en: https://goo.gl/v8cUiB [Accedido: 18-jun-2017]

[41] Biblioteca de la Universidad Pública de Navarra. Oficina de Referencia, «Guía para citar y referenciar. APA Style», 2014. [En línea]. Disponible en: http://goo.gl/0CSj5G [Accedido: 16-abr-2017]

[42] “Stack Overflow”, *Stack Exchange Inc.*, 2017. [En línea]. Disponible en: https://stackoverflow.com/ [Accedido: 17-jun-2017]

[43] C.G. Pere, “SQL Fácil”, *Editorial Marcombo*, 2014.

[44] “Java: JDBC”, lista de videos tutoriales, *R.Daniel “Makigas”,* 2017 [En línea]. Disponible en: https://www.makigas.es/ y canal de Youtube: https://goo.gl/lMcki9 [Accedido: 12-abr-2017]

[45] “Google URL Shortener”, *Google,* 2017. [En línea]. Disponible en: https://goo.gl/. [Accedido: 16-abr-2017]

[46] “Controlador JDBC 6.0 de Microsoft para SQL Server”, *Microsoft*, 2017. [En línea]. Disponible en: https://goo.gl/6C7wKh [Accedido: 13-jun-2017]

[47] “Download Swingx.jar”, *Demo Source and Support*, 2012. [En línea]. Disponible en: https://goo.gl/TGirNi [Accedido: 14-jun-2017]

[48] M. Ross, M. Stacia, “Introducing Microsoft SQL Server 2012”, *Microsoft Press*, 2012.

[49] E. Bruce, “Thinking in Java, 4th Edition”, *Prentice Hall*, 2006.

[50] “JDBC frente a ODBC y otros API’s”, *Carmen Arvelo* en *Scribd*, 2017. [En línea] Disponible en: https://goo.gl/tNRSc0 [Accedido: 15-jun-2017]

[51] “Java Platform, Standard Edition 7 API Specification”, *Oracle*, 2017. [En línea] Disponible en https://docs.oracle.com/javase/7/docs/api/ [Accedido: 18-jun-2017]

ANEXO A: Glosario de términos

Se justifica la presencia de este anexo puesto que a lo largo de la memoria se tratan conceptos técnicos relativos a los amortiguadores y su mecánica.

- **Referencia**: nombre único que distingue a un amortiguador.
  Ejemplo: el amortiguador con referencia 4729 corresponde a las ruedas traseras del modelo *Fiesta* de la casa *Ford*.

- **Reglaje**: la Real Academia de la lengua Española (RAE) define reglaje en su acepción mecánica como: *el reajuste de las piezas de un mecanismo para mantenerlo en su correcto juego de funcionamiento* [15].
  En la memoria se emplea la palabra “reglaje” como un conjunto de medidas que rigen el ensayo de un amortiguador: equivaldría el reajuste de la máquina que ejecuta el ensayo.
  El reglaje será distinto en función del amortiguador que se quiere testear. Un reglaje tiene asociados datos como la velocidad o el tipo de señal que imprime.
  Los datos del reglaje se dividen en dos grupos: las precargas, y las condiciones.

- **Precarga**: es el dato de fuerza mínimo para el cual un sistema de suspensión (la finalidad del amortiguador), se activa [16], [17], [18] .En este proyecto las precargas son una parte del reglaje.

- **Condiciones**: normalmente no se ejecuta una sola prueba sobre el amortiguador. Se denominan condiciones a las repeticiones del ensayo. Cada una transmite un valor distinto de velocidad, amplitud de señal, tipo de señal, etc. a la máquina que hace el ensayo. En este proyecto las condiciones son parte del reglaje.

- **Equivalencia**: grupo de referencias que comparten el mismo reglaje.
  Las referencias se separan con barras. Ejemplo: la equivalencia *6217/6218* agrupa dos referencias que comparten reglaje, la referencia *6217* y la referencia *6218*.

- **Cargas de compresión / tracción**: fuerza, medida generalmente en daN (unidad de 10 Newtons), que un amortiguador experimenta cuando la máquina de ensayo lo empuja (compresión), o lo estira (tracción). Es fundamental para determinar si el amortiguador es defectuoso o no. Por ello, en las condiciones del reglaje se definen intervalos de esta medida, mediante límites inferiores y superiores en ambas cargas.

ANEXO B: Tecnologías

Las tecnologías usadas a lo largo de este proyecto han sido:

- Artifactory versión 4.13.0
- Dispositivo hardware, Data Adquisition Device: modelo NIUSB-6002, de National Instruments
- GitLab Community Edition 8.12.4
- Gradle versión 0.1
- Plugin de soporte de Gradle en Netbeans, versión 1.4.1
- IDE Netbeans 8.2
- Java versión 1.8.0
- Controlador Microsoft JDBC para SQL Server, versión 6.0
- Notepad++ versión 7.4.1
- Máquina virtual: Oracle VirtualBox versión 5.1.18
- Paquete Microsoft Office 2003, del cual:

    - Microsoft Access 2003

- Paquete Microsoft Office 2016, del cual:
    - Microsoft Access 2016
    - Microsoft Excel 2016

- Paquete Microsoft SQL Server 2012 Express, del cual:
    - Microsoft SQL Server 2012 Express
    - Microsoft SQL Server Management Studio 2012

- SourceTree versión 2.0.20.1

- Librería SwingX versión 1.6.4

- Wekan versión 0.23

Todo el software usado ha sido de libre distribución y gratuito, a excepción del paquete Microsoft Office. El cual se obtiene gracias a la versión para estudiantes contratada por la Universidad Pública de Navarra.

Los accesorios y servidores de la intranet pertenecen a Engineea.

ANEXO C: Conocimiento técnico adquirido

Este anexo pretende informar sobre el aprendizaje de las herramientas y soluciones necesarias en el proyecto.

Gracias a este semestre de prácticas empresariales la autora ha ampliado su conocimiento sobre desarrollo software, metodologías ágiles y herramientas.

C.1 Intranet

En equipos de desarrollo software es muy común el uso de una intranet: una red privada para la empresa, sólo accesible dentro de la misma.

Dentro de esta intranet se tienen recursos y servicios compartidos, como los siguientes.

1. **Metodología de tablero Kanban: software Wekan**

Wekan es una herramienta que sigue la metodología *kanban* y que, según su equipo, es la versión gratuita y de código abierto del famoso Trello [19].

Wikipedia define el *kanban* como [20]:

[…]Un sistema de información que controla de modo armónico la fabricación de los productos necesarios en la cantidad y tiempo necesarios en cada uno de los procesos que tienen lugar tanto en el interior de la fábrica, como entre distintas empresas. […]

Es un subsistema del JIT (Just-In-Time).

El equipo creador de Kanban Tool [21] (otra herramienta muy semejante a Wekan), escribe en su web:

La metodología Kanban […] es capaz de gestionar el trabajo de forma fluida. Proveniente de Japón, Kanban es un símbolo visual que se utiliza para desencadenar una acción. A menudo se representa en un tablero Kanban para reflejar los procesos de su flujo de trabajo.

[…] La tarjeta Kanban se moverá a través de las diversas etapas de su trabajo hasta su finalización. A menudo se habla de él como un método de extracción, de forma que usted tira de sus tareas a través de su flujo de trabajo, ya que permite a los usuarios mover de sitio libremente las tareas en un entorno de trabajo basado en el equipo.

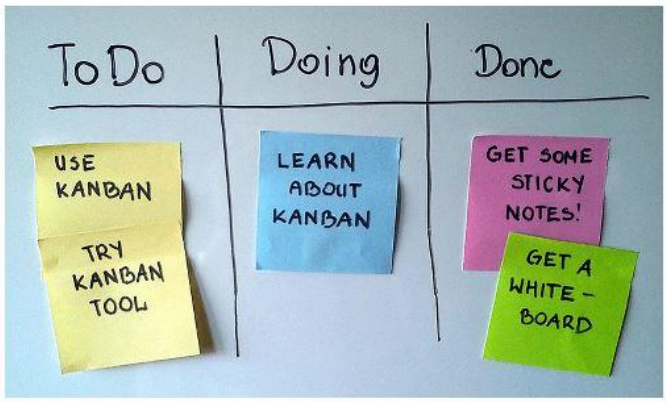


Ilustración 93: tablero kanban en la web Kanban Tool

Wekan tiene muchas funcionalidades, y a lo largo del proyecto, me ha servido como un organizador personal y de equipo; manteniendo informado al tutor del estado del proyecto y las tareas que estaba realizando.

Posee un tablero en el cual se pueden incorporar miembros al equipo, para poder acceder a él, editarlo, etc. En definitiva, hacer un proyecto colaborativo.

Se pueden crear tarjetas con su descripción, donde marcar listas, puntos de control (checkpoints), tareas, etc.

Los colaboradores pueden dejar comentarios, y existe un registro de la actividad, para supervisar los últimos cambios hechos en el tablero.

Se puede organizar por diferentes columnas y añadir etiquetas (colores) con distintos significados.

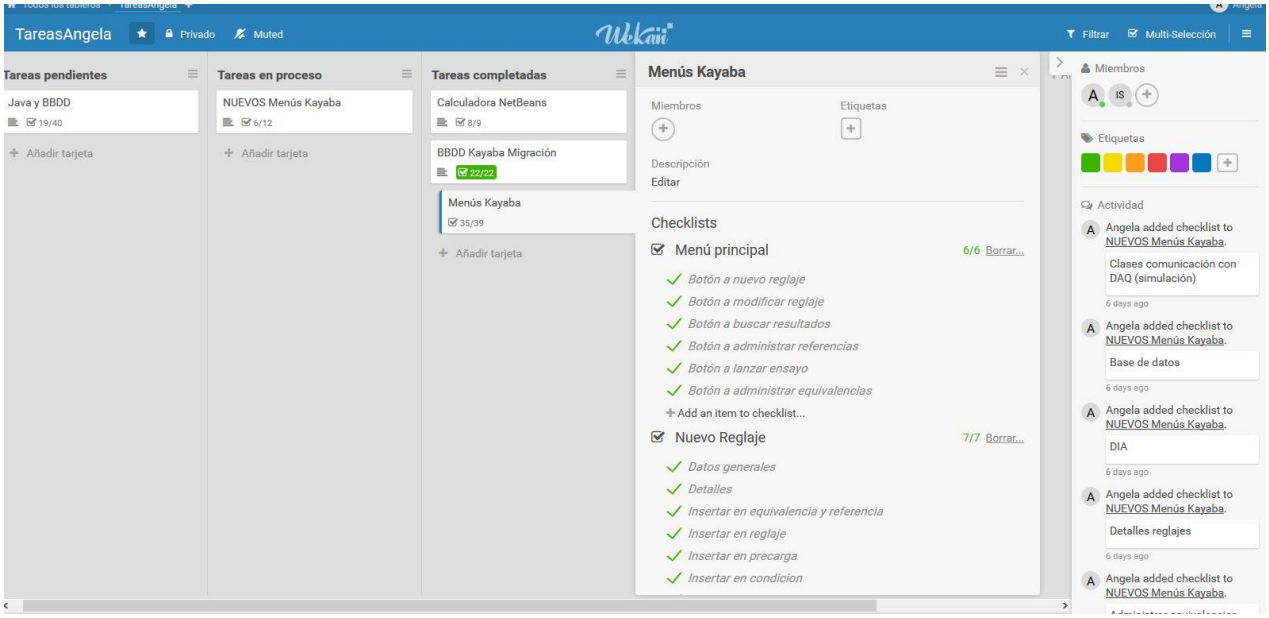


Ilustración 94: interfaz online de Wekan

2. Control de versiones: Gitlab y SourceTree

Los sistemas de control de versiones son un tipo de herramientas software que ayudan a los equipos de desarrollo a gestionar los cambios en el código fuente […].

Hacen seguimiento de cada modificación en el código en un tipo especial de base de datos. Si se comete un fallo, los desarrolladores pueden volver atrás y comparar con versiones anteriores del código para reparar el error minimizando la disrupción con el resto de miembros del equipo. – Traducción [22].

Git define de esta forma los sistemas de control de versiones.

Git es el sistema de control de versiones más utilizado en la actualidad. Se desarrolló en 2005 por Linus Torvalds, creador del kernel del sistema operativo Linux.

Tiene una arquitectura distribuida: cada miembro del equipo desarrollador tiene un repositorio propio, local, que contiene toda la historia de cambios hechos en el código. [23]

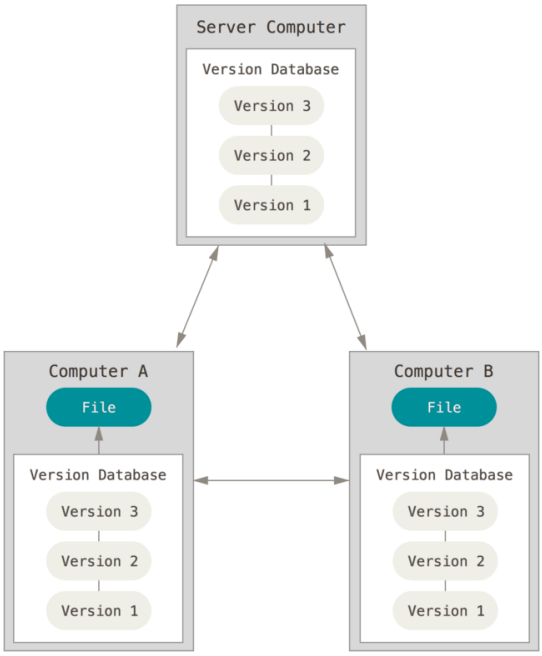


Ilustración 95: arquitectura distribuida de control de versiones

Gitlab es un gestor online de repositorios Git, [24]

SourceTree es una aplicación de escritorio con el mismo fin. [25]

Algunos conceptos de interés en los sistemas de control de versiones son:

- **Repositorio**: es el lugar donde se almacena la historia de versiones de código.
- **Stage file**: se preparan (notifican) los cambios que se han hecho en ficheros de código.
- **Commit**: consiste en publicar estos cambios (sólo en los ficheros marcados como staged), en el repositorio local.
- **Push**: aplica los cambios del commit en el repositorio disponible online.
- **Pull**: aplica los cambios del repositorio accesible online, al local.
- **Branch**: puesto que los miembros del equipo pueden editar los mismos ficheros, se deben de crear "ramas" diferentes de trabajo (branches), como un árbol.

  La rama principal y la definitiva, que hace el símil con el tronco del árbol, es la denominada "master".
- **Merge**: converge ramas de trabajo distintas.

- **Conflictos**: los conflictos son fallos al intentar acciones como merge, push,pull … que hacen al sistema incapaz de fusionar cambios. Es necesarios resolverlos: editando los ficheros, combinando, o seleccionando la versión adecuada.

  Un ejemplo de conflicto ocurre si un usuario edita una línea de código en la rama principal; sube el cambio, y otro usuario sin tener noción del cambio (no hace pull), edita la misma línea y quiere hacer push en el repositorio. Existe un conflicto de versiones que tiene que ser resuelto.

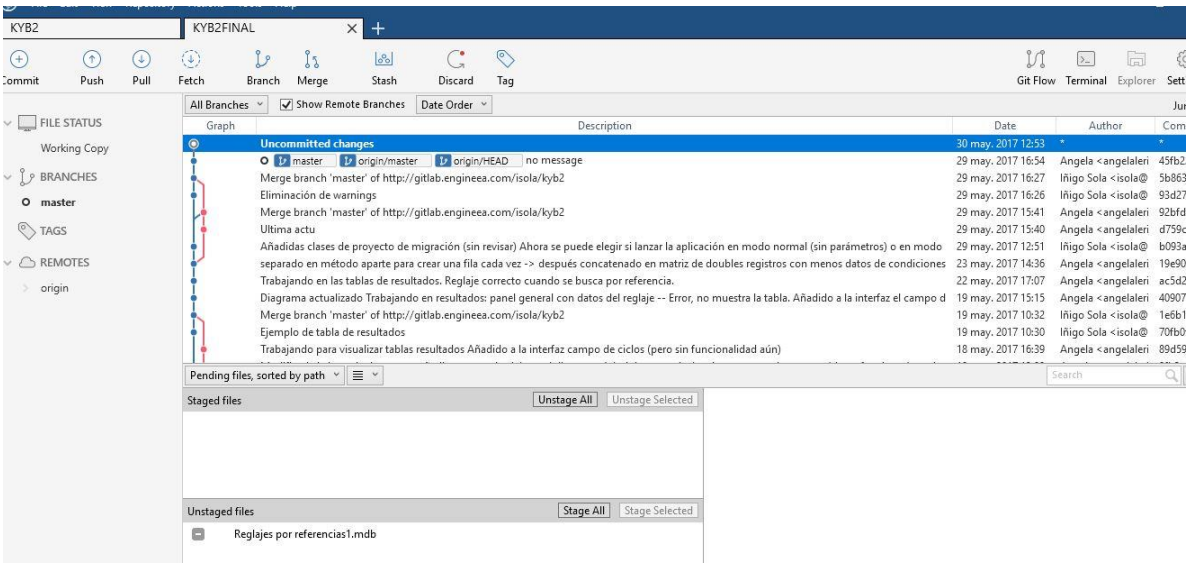


Ilustración 96: Captura del software SourceTree. Ramas, merge de ramas, stage files...

En los proyectos de desarrollo de software, algunos programas como los entornos IDE (Netbeans) o herramientas de automatización (Gradle), generan ciertos archivos que no interesan en el control de versiones.

Se puede crear un fichero *.gitignore* y marcarlo (stage) en el repositorio, de forma que ignore los archivos indeseados. Existen herramientas online que crean este fichero sin más que añadir el nombre de los programas [26].

Ilustración 97: web para crear archivo .gitignore

```
# Created by https://www.gitignore.io/api/gradle,netbeans

### NetBeans ###
nbproject/private/
build/
nbbuild/
dist/
nbdist/
.nb-gradle/

### Gradle ###
.gradle
/build/

# Ignore Gradle GUI config
gradle-app.setting

# Avoid ignoring Gradle wrapper jar file (.jar files are usually ignored)
!gradle-wrapper.jar

# Cache of project
.gradletasknamecache

# # Work around https://youtrack.jetbrains.com/issue/IDEA-116898
# gradle/wrapper/gradle-wrapper.properties

# End of https://www.gitignore.io/api/gradle,netbeans
```

Ilustración 98: fichero .gitignore generado a través de la web

3. Herramientas de automatización de desarrollo software (build automation tools)

En el desarrollo software es muy empleada la automatización de procesos, que ahorran tiempo y esfuerzo en tareas comunes.

Para ello, existen una serie de herramientas, como Gradle [27].

Entre otras virtudes, con Gradle es posible ejecutar un proyecto de Java que utilice ciertas librerías sin tenerlas almacenadas en la máquina.

En un proyecto Gradle se definen las librerías u otros elementos necesarios en un archivo (script) de configuración.

Al ejecutar el proyecto, las busca en los repositorios especificados y las descarga en el momento.

Otra famosa herramienta de construcción es Maven [28].

Existe repositorios online abiertos a todo el mundo, y repositorios de uso privado.

La empresa cuenta en su servidor con Artifactory [29], que además de incluir librerías de uso común extendido, puede almacenar código propio creado en la empresa. De este modo el equipo puede compartir su trabajo fácilmente dentro de la compañía.

C.2 Máquina virtual

Para poder acceder a la base de datos, se contó con una máquina virtual.

Una máquina virtual es un software que emula a un ordenador como si fuese una aplicación del propio ordenador físico [30]. Ello le permite tener otros sistemas operativos distintos al ordenador en el que se ejecuta. Por ejemplo, Ubuntu o en el caso del proyecto: Windows XP.

Existen muchas máquinas virtuales, pero se ha elegido usar VirtualBox, de Oracle [31] debido a su sencillez y extendido uso.

Tras la correcta instalación del sistema operativo, se descargaron las "Guest Additions" de VirtualBox: un paquete con una serie de controladores que mejoran el manejo de la máquina virtual [32]. Entre ellos se destaca la fluidez del paso del ratón entre máquina virtual – real, añadido a una mejora en el comportamiento de video.

En esta máquina virtual se instaló Microsoft Access 2003 para poder abrir y analizar la base de datos original.

Se enlazó una carpeta de la máquina virtual con la real para poder traspasar los archivos. Se evidencia en la parte inferior de la siguiente captura.

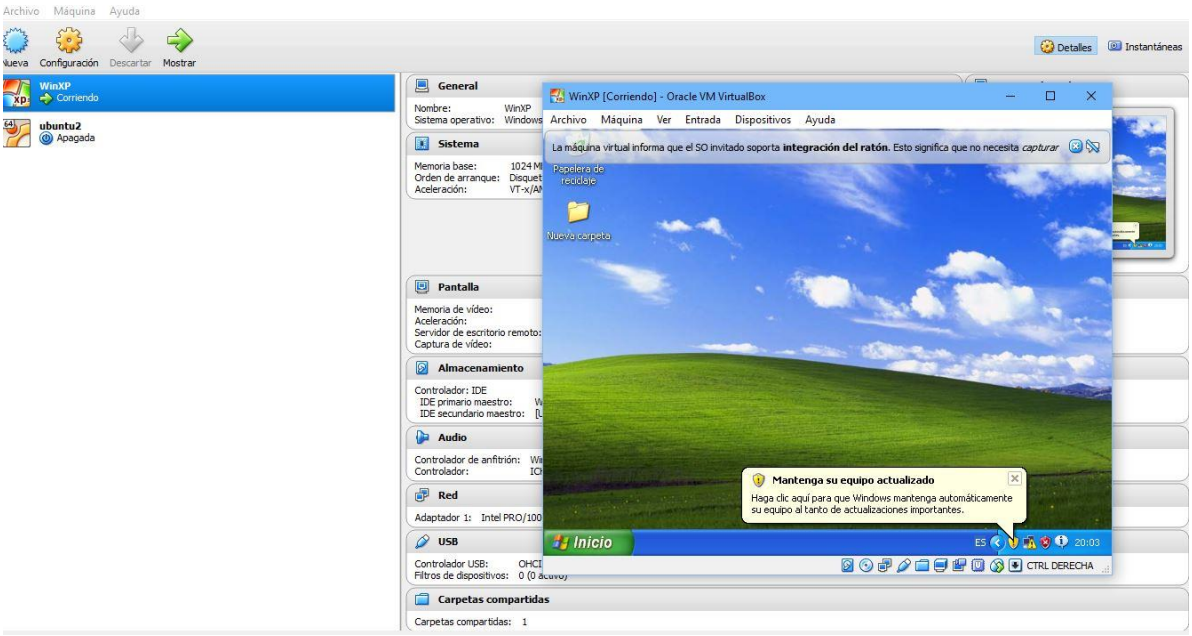


Ilustración 99: máquina virutal de Windows XP con VirtualBox de Oracle

C.3 Software de diagramas DIA

El software DIA [33] es un programa informático que ofrece de forma muy simple realizar diagramas. Desde DIA se pueden crear recuadros que simbolizan a las tablas de una base de datos, añadir los atributos y especificar el tipo de datos que almacenan.

Se pueden añadir colores a las tablas, títulos, notas, relaciones, etc.

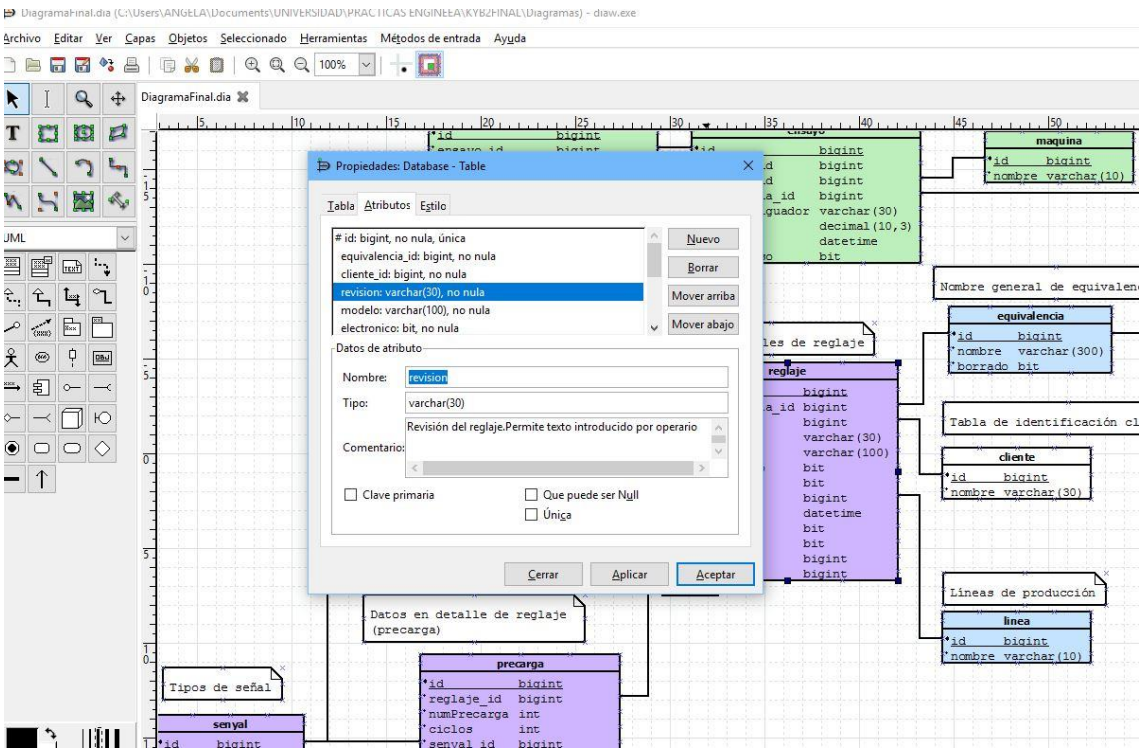


Ilustración 100: Ejemplo de edición en DIA

C.4 Bases de datos

Antes de comenzar el proyecto se tenían unos conocimientos básicos sobre bases de datos y lenguaje de consulta gracias a algunas asignaturas cursadas en el grado.

Sin embargo, se ha profundizado más en el uso de éste, los fundamentos teóricos de bases de datos, software gestor de bases de datos, etc.

Algunos de estos conceptos descubiertos son los siguientes.

1. **Modelo relacional de base de datos**

Una base de datos relacional es un conjunto de tablas, compuestas por filas y campos, que se definen por sus relaciones.

Es el modelo más usado de base de datos por ser sencillo y comprensible, aunque existen otros: en red, jerárquico, en estrella…

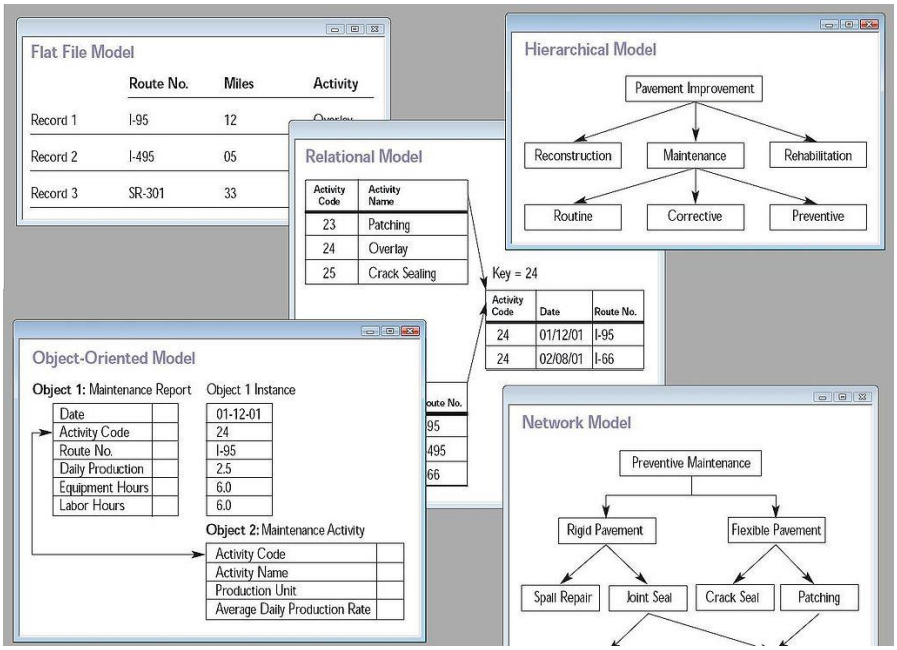


Ilustración 101: distintos modelos de bases de datos

Las bases de datos son administradas por los SGBD: Software/Sistemas Gestores de Base de Datos o Relational Database Management Systems (RDBMS).

SQL Server, MySQL y Oracle Database son SGBD.

En este proyecto se usa SQL Server por petición de cliente. Aunque éste envió la base de datos actual de la fábrica en formato de Access.

Access supone algunos inconvenientes: genera tablas de errores y no permite las mismas consultas SQL que SQL Server: tiene su propia sintaxis en algunas cláusulas.

SQL Server Management Studio tiene una interfaz gráfica ordenada y de fácil uso. Es muy versátil y lo hace más atractivo que Access; aunque su fin sea el mismo.

MySQL también es un SGBD muy cómodo, que incluye su propio software gráfico. Tiene algunas diferencias respecto a SQL Server, por ejemplo, en la escritura de algunas consultas.

Se empleó al inicio del proyecto a modo de práctica de lenguaje SQL, conexiones, etc.

2. Consultas, lenguaje SQL

El Lenguaje de Consulta Estructurado (Structured Query Language, SQL), es un lenguaje de programación que gestiona bases de datos relacionales.

Wikipedia [34] destaca sus características en: *el manejo del álgebra y el cálculo relacional que permiten efectuar consultas con el fin de recuperar, de forma sencilla, información de bases de datos, así como hacer cambios en ellas.*

Se tenía un conocimiento básico sobre este lenguaje, que se ha ampliado por la necesidad de frecuentes consultas y operaciones con la base de datos. Todas ellas vienen bien detalladas en la página web de W3Schools [35].

Además de vistas y uniones, se ha aprendido a hacer pequeñas subconsultas.

Esto es, realizar una consulta dentro de otra. Ha sido necesario para crear la vista de referencias únicas, ya que se hace una consulta agrupando y contando las referencias por su equivalencia, y después seleccionando aquellos grupos cuyo conteo daba igual a 1.

3. Vistas

Microsoft [36] define las vistas como: *una tabla virtual cuyo contenido está definido por una consulta.*

En el proyecto se ha creado una vista llamada *referenciasUnicas*.

Su principal objetivo es reducir la escritura de código en las clases Java, haciendo las consultas más cortas. Es muy útil si se tienen bases de datos como la del proyecto que contienen tablas con muchas interrelaciones.

4. Funciones de agregado

Una función de agregado consiste en un cálculo sobre un conjunto de valores, que devuelve un único dato [37].

Existen diversas funciones de agregado:

- SUM: realiza un sumatorio
- AVG: calcula la media aritmética
- COUNT: calcula un recuento
- MIN: devuelve el valor mínimo
- Etc.

Al hacer un cálculo sobre un conjunto de datos, no es posible devolver o seleccionar datos que estén internos a este interno. Por ejemplo, si agrupamos referencias por su equivalencia, no es posible seleccionar si están borradas o no; puesto que es algo propio de cada una de las referencias y no del grupo.

Una de las funciones de agregado utilizadas en el proyecto es precisamente esta, la de la agrupación de referencias por equivalencia.

5. Uniones

Una unión (o join) es la combinación de tablas mediante columnas de ellas que mantienen alguna relación.

Hay muchos tipos de joins, y recuerdan a las teorías de conjuntos [38].

En el proyecto se hacen uniones simples mediante las relaciones de identificación, para poder obtener en una sola consulta datos de dos tablas de la base de datos nueva.

En el caso de la base de datos original, ha sido necesario unir tablas para conseguir todos los registros, por ejemplo, de las referencias. Ya que aparecían distintas referencias en ambas tablas.

El siguiente diagrama se ha utilizado a lo largo del proyecto como base teórica:

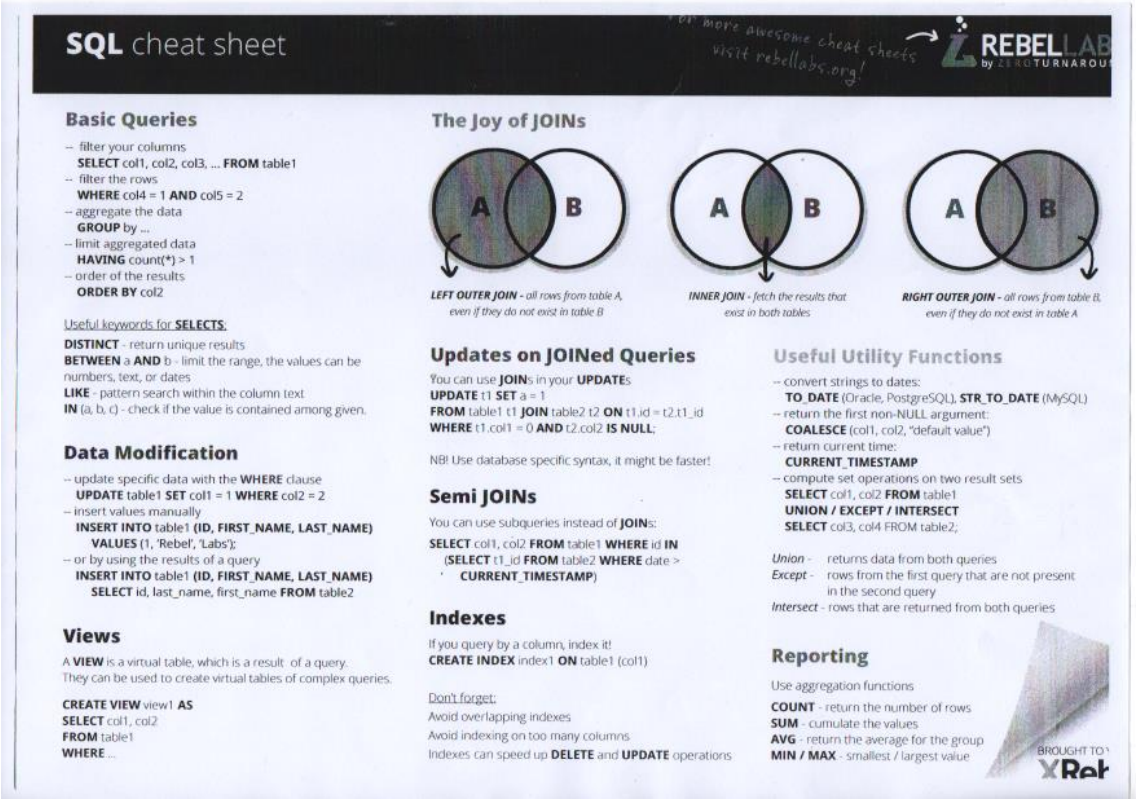


Ilustración 102: diagrama de uniones en SQL

6. Triggers

Microsoft define los desencadenadores (o triggers) como [39]: *un procedimiento almacenado que se ejecuta automáticamente cuando ocurre un evento en el servidor de la base de datos.*

En el proyecto se ha escrito el código en SQL, para un trigger que se activa cuando se inserta un registro en la tabla *reglaje* de la base de datos nueva.

Su funcionamiento es:

1) Insertar la fila
2) Comprobar si el campo “reglaje_id” es nulo.
3) En caso afirmativo, tomar el identificador generado, y colocarlo en dicho campo.

7. Generación de scripts para creación de las bases de datos

El software SQL Server Management Studio facilita el traspaso de bases de datos gracias a una herramienta que genera scripts (ficheros) que automáticamente crean bases de datos [40].

Dentro de estos ficheros, están las líneas de código necesarias para crear el esquema y tablas de la base de datos deseada. Así como vistas, relaciones, diagramas de relación… todas las opciones que se deseen especificar. También el traspaso de datos contenidos en una base.

Hay que prestar especial atención a estas opciones, y a la diferencia de la generación del script según versiones del servidor SQL. Por ejemplo, en el proyecto ocurre que la versión 2008 no ofrece las mismas opciones de script que la de 2012 (en la que se trabaja).

C.5 Competencias informacionales

El cursillo de competencias informacionales de la Universidad Pública de Navarra [41] ha sido muy útil para redactar esta memoria y buscar la información competente.

Se intentó en un inicio usar el programa Mendeley para las referencias bibliográficas. Es un software de gestión de referencias muy completo, pero muy estricto.

Mucha información sobre programación y desarrollo se encuentra online; además de ser cuestiones concretas que profesionales por libre comparten y resuelven en foros.

Sin embargo, es importante hacer un buen uso de la información en internet y foros conocidos entre desarrolladores, como Stack Overflow [42].

En él, los usuarios plantean dudas que deben explicar con claridad y acompañar con ejemplos.

A menudo un mismo problema tiene múltiples respuestas, por lo que la lectura es muy provechosa. Es norma en el foro sostener las respuestas con documentación oficial, como la API de Java, documentación de Microsoft SQL Server, etc.

El software Mendeley parece más adecuado para documentos de investigación científica o de nivel superior al grado, donde se consultan artículos de seminarios, revistas, tesis, etc.

Por ello, se decide seguir el estándar IEEE y escribir en marcadores las referencias bibliográficas. Luego, al redactar la memoria, se insertan “referencias cruzadas”, que introducen el número de marcador en el texto (ejemplo: [1]), enlazado correctamente a su apartado en la Bibliografía.

ANEXO D: Relación con el plan de estudios

D.1 Introducción

Se consideró útil añadir a la memoria un apartado donde se explicitase la relación de los conocimientos enseñados en el grado de Ingeniería en Tecnologías de Telecomunicación,

cursado en la Universidad Pública de Navarra, con la realización del proyecto y de las prácticas en empresa.

Se agrupan en cursos aquellas asignaturas que han aportado conocimiento relacionado con la temática del trabajo.

D.2 Primer curso

D.2.1 Informática

Access: En esta asignatura se introdujo de forma breve el concepto de base de datos. Se realizaron consultas sencillas a través de la interfaz gráfica.

Programación: formación en Pascal. Es una buena base para empezar a entender la lógica de programación en su nivel más simple y secuencial. El lenguaje es intuitivo y se enseña el uso de bucles, variables, etc. Sin embargo, limitado, tiene fallos recurrentes en versiones y no tiene uso real hoy en día.

D.3 Segundo curso

D.3.1 Arquitectura de Redes, Sistemas y Servicios

Lenguaje Java: se empieza a introducir Java de manera práctica.

Falta una base teórica en lenguajes de programación orientado a objetos (OOL): conceptos de métodos, constructores, abstracciones, interfaces, excepciones, herencia…

Se centra en la resolución de problemas matemáticos, como la función B de Erlang.

D.3.2 Redes de Ordenadores

El temario de esta asignatura contenía conceptos de sockets, paquetes de transporte TCP, UDP, diseño de redes, visualización de paquetes IP a través de la red (software Wireshark) … En ocasiones es útil comprobar el estado de una conexión, por ejemplo, al servidor donde se aloja una base de datos, entendiendo los paquetes que se envían al hacer ping, ver los paquetes ICMP de error…etc.

También se hizo uso de Java en esta asignatura.

D.4 Tercer curso

D.4.1 Fundamentos de Tecnologías y Protocolos de Red

Supone una continuación de la asignatura Redes de ordenadores. Muy útil en la comprensión del diseño de redes, estado de las conexiones, puertos, etc.

D.4.2 Servidor web: servidor

MySQL, PHP y PHPMyAdmin: se crea una página web como proyecto de la asignatura. Se dan conocimientos básicos de bases de datos.Se ven algunas uniones de tablas muy simples. Es muy positivo el uso de PHP y PHPMyAdmin. Aunque su uso (PHPMyAdmin) evita la escritura de consultas SQL.

Esta asignatura ha sido fundamental para el proyecto en el tema de bases de datos y lenguaje de consultas.

D.4.3 Laboratorio de programación

Lenguaje Java: en esta asignatura se asienta el conocimiento de Java.

Se centra más en el graficado pixel a pixel, desarrollando pequeños videojuegos y usando eventos de teclado.

No se incluyen conceptos abstractos como buenas prácticas de escritura de código: modelo vista controlador, control de versiones, repositorios, compactación de código escrito, estructuración en paquetes, adición de librerías, conexión desde Java a bases de datos… En la asignatura no se presentan las GUI (Interfaces Gráficas de usuario); que cualquier programa utiliza y a lo que muchos desarrolladores se dedican. Sería muy positiva la inclusión de estos conceptos.

D.5 Durante el grado

Máquinas virtuales: a pesar de no estar relacionado directamente con el proyecto, a menudo en el mundo de las tecnologías de la información, se necesitan otros sistemas operativos y/o versiones, para ejecutar operaciones no disponibles en nuestro sistema habitual. Por ello, se acude a las máquinas virtuales. En el grado, existen asignaturas donde esto es necesario. Por ejemplo, en todas las asignaturas del área de telemática, se ha empleado sin excepción el sistema Ubuntu; puesto que la configuración de routers y redes a través de la consola es más cómoda que en Windows.

En la asignatura de Televisión Digital Interactiva, se utiliza un paquete de difusión de señal de televisión (Opencaster) que solo está disponible para Linux.