Facultad de Ciencias

Aplicación para la confección de calendarios

Timetabling application

Trabajo de fin de grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Manuel González Iglesias
Codirector: Domingo Gómez Pérez
Codirectora: Beatriz Porras Pomares

Junio 2018

Agradecimientos

A mi familia y amigos, porque siempre están ahí.
A mis tutores, Domingó Gómez y Beatriz Porras, por su apoyo y colaboración durante todo este proceso.

Aplicación para la confección de calendarios

Resumen

palabras clave: aplicación, gestión de horarios, datos, base de datos

La creación y modificación de horarios en una institución docente supone un problema mucho más complejo de lo que pueda parecer a simple vista. La dificultad radica en la necesidad de satisfacer gran cantidad de restricciones administrando recursos de forma efectiva. Esta complejidad se extiende a la cuestión de la representación y almacenamiento de los datos.

Este proyecto se centrará en la gestión efectiva de la información, incluyendo su obtención, representación, almacenamiento y recuperación. Se construirá una base de datos apropiada para este fin.

Además, partiendo de una aplicación preexistente se generarán métodos de acceso a esos datos. La aplicación accederá a la base de datos para visualizar, insertar o modificar datos, manteniendo la coherencia en todo momento.

Timetabling application

Abstract

keywords: application, timetable scheduling, data, database

The creation and management of schedules for a higher education institution constitutes a greater problem than it may appear. This difficulty is rooted on the need to satisfy a variety of restrictions by effectively managing resources. Such complexity extends to the mater of data storage and representation.

This project will be centered on the effective administration of information, including its representation, storage and retrieval. An appropriate database will be built to that end.

Additionally, we will generate data access methods for a preexisting application. This application will access the database in order to visualize, insert or modify data, maintaining coherence through the process.

Índice

1 Introducción

En una institución como la Universidad de Cantabria la creación de horarios constituye un problema de satisfacción de restricciones y gestión de recursos.

En esta universidad, en concreto, se han de coordinar los horarios de cuatro grados distintos (además de varios programas de máster que no se han contemplado). Para cada uno de esos grados debemos considerar cuatro cursos, cada uno con diferentes asignaturas, que varían en número, créditos, horas de teoría, problemas y práctica, entre otros parámetros.

Esto se traduce en una serie de eventos (horas de clase) que deben impartirse semanalmente.

Para que un evento pueda celebrarse en determinado espacio de tiempo, se requiere dedicar recursos: necesitaremos un aula con las características apropiadas (o un laboratorio, como mínimo), un grupo de alumnos (dependiendo de nuestro planteamiento del problema probablemente serán más) y al menos un profesor (en horas de prácticas, más).

Actualmente en nuestra facultad, este conjunto de circunstancias debe ser contemplado y resuelto por una sola persona. No sólo deberá considerar esos parámetros, que parecen relativamente obvios, también deberá atender a otras restricciones que presenta la realidad de la situación.

Esto incluye, por ejemplo, intentar que las horas de clase de los distintos grupos de alumnos tengan lugar por la mañana en la medida de lo posible, que se concentren de lunes a jueves, que las tardes se dediquen a prácticas de laboratorio, que queden el menor número posible de horas libres entre clases, que las horas de clase de los profesores se concentren al principio o final de la mañana...

Como podemos ver, esto supone un conjunto enorme de datos interrelacionados y restricciones. Sea una persona o un programa quien deba realizarlo, la búsqueda de la solución nunca será tarea fácil.

Existen actualmente distintas propuestas que pretenden dar resolver este problema. Uno de ellos es *XHSTT*, un *framework* de representación de datos especializado para esto.

Este proyecto continuará con uno del año pasado [Vejo Gutiérrez, 2017] que, basándose en *XHSTT*, generó una aplicación de gestión de horarios para la Universidad. En este caso, me centraré en mejorar esa aplicación, añadiendo nuevos módulos y funcionalidades, persiguiendo el mismo objetivo de facilitar la labor de creación de horarios.

Estructura de la memoria

Los siguientes capítulos recogerán las distintas partes y aspectos del proyecto:

El capítulo 2 describe la situación de la que partimos en este proyecto, considerando la aplicación desarrollada en el Trabajo previo y las necesidades heredadas y nuevas.

El capítulo 3 se centra en el proceso de diseño y creación de una base de datos adaptada a nuestro problema.

El capítulo 4 explica cómo se han introducido los datos en las tablas, desde qué fuentes y con qué procesos.

El capítulo 5 pasa de la base de datos a los mecanismos mediante los que esta se integrará en la aplicación preexistente.

El capítulo 6 trata el proceso de implantación del sistema, los desafíos que eso plantea y las nuevas necesidades que surgen.

El capítulo 7 cierra el proyecto formulando las conclusiones que se derivan del mismo y las posibilidades que ofrece el futuro de la aplicación.

Cierran la memoria un apéndice (A), con instrucciones para el uso de la aplicación, y la sección de referencia bibliográficas.

2 Evaluación del proyecto

2.1. Situación inicial

El punto de partida de este proyecto es la necesidad de continuar añadiendo funcionalidad al proyecto *'Desarrollo de una Aplicación para la Planificación de Horarios'* [Vejo Gutiérrez, 2017], presentado en la convocatoria de Septiembre de 2017 por el alumno José Ramón Vejo Gutiérrez como su propio Trabajo de fin de Grado.

Ese proyecto consistió en la elaboración de una aplicación de gestión de horarios. Los principales elementos y herramientas del proyecto son los siguientes:

- ***Lenguaje de programación:*** El código de la aplicación está elaborado en lenguaje *Python 3*. La elección de este lenguaje en concreto atiende al objetivo de portabilidad y legibilidad del proyecto. Está gestionado por la *Python Software Foundation* [1].

  Consecuentemente, en este nuevo proyecto heredaremos esos objetivos.

- ***Entorno de usuario:*** La interfaz se elaboró utilizando *Kivy* [5], una librería de *Python* que permite elaborar interfaces de usuario con relativa facilidad y versatilidad. La interfaz generada es visualmente sencilla, pero al mismo tiempo funcional e intuitiva.

- ***Capa de datos:*** El almacenamiento de datos de la aplicación utiliza el formato *XHSTT* [2]. Se trata de un formato de representación de datos basado en *XML* que permite formular problemas de gestión de horarios escolares. El archivo que se almacena contiene el planteamiento completo del problema, incluyendo todos los tiempos, eventos, recursos y restricciones del problema. Además, en caso de que se elabore una solución, está se almacena como continuación en el mismo archivo.

  El formato *XHSTT* favorece la visualización del problema, siendo la legibilidad una característica esencial del paradigma *XML*.

  Como contrapartida, no se trata de un sistema de representación y almacenamiento de información que garantice la coherencia de los datos por sí mismo. No impone restricciones estrictas en la introducción o modificación del modelo. Resolver este problema será el objetivo principal del presente proyecto.

- ***Software adicional:*** Por un lado, se utiliza la librería open-source *KHE* [3], que permite generar soluciones a planteamientos de problemas de timetabling planteados correctamente en formato *XHSTT*.

  Por otro lado, una vez se generan soluciones con *KHE*, se hace uso de la herramienta online *HSEval* [4] para evaluación de posibles soluciones de problemas de horarios escolares en *XHSTT*.

En este nuevo proyecto no se hará un uso directo de todas estas tecnologías pero será necesario considerarlas para integrarlo con el original.

2.2. Objetivos y Tecnología

Este nuevo proyecto tiene como principal objetivo incorporar a la aplicación un sistema de almacenamiento de datos formal que asegure en cierta medida la coherencia de los datos almacenados y que permita la introducción, modificación, visualización y/o eliminación de datos de forma estructurada.

Si bien *XHSTT* ofrece una visualización completa del problema, que se incluye completo en un archivo, y cuenta con una estructura relativamente legible (heredada de *XML*), no se trata de un formato utilizado convencionalmente. Esto presenta una barrera de entrada que dificulta la implantación de un sistema basado en un sistema así en un contexto en el que pueden coexistir usuarios especializados e inexpertos.

Trabajamos pensando en las futuras posibilidades y necesidades de nuestro sistema.

En primer lugar, es de esperar que cada cuatrimestre se produzcan cambios en el problema que se debe plantear y resolver. En una institución como esta Universidad, cada año es necesario plantear al menos dos problemas de horarios; uno por cada cuatrimestre, cada cual con distinto conjunto de eventos a los que asignar tiempos y recursos. No sería extraño tampoco que en un mismo cuatrimestre hubiese que reformular y volver a resolver el problema debido a cambios imprevistos.

A esto se suman los cambios que se producen de forma natural en un centro docente a medida que pasa el tiempo. La plantilla de personal docente y los grupos de alumnos cambian cada año; el currículum de una carrera evoluciona para reflejar la demanda o los avances en los campos de conocimiento relevantes a los grados; las obras en las infraestructuras pueden determinar cambios en los recursos disponibles, pudiendo por ejemplo añadir o ampliar aulas, o restringir el uso de estas temporalmente.

En segundo lugar, actualmente los horarios son elaborados por el jefe de estudios de cada facultad; sin embargo, en la Era de la Información podemos esperar que el paradigma evolucione. El proyecto ha sido concebido teniendo presente la posibilidad de que, ya que un horario escolar debe coordinar una plantilla relativamente extensa, la responsabilidad de su elaboración se reparta entre múltiples personas.

Incluso si la organización de la institución no cambiase, un sistema accesible a varios usuarios con diferentes funciones permitiría generar un sistema basado en un usuario con privilegios muy restringidos que pusiese a disposición de todos los alumnos y miembros del personal (a travçes de algún sistema como la secretaría virtual de la universidad o la plataforma *Moodle*) búsquedas y visualizaciones de horarios personalizadas para sus necesidades. Podrían generarse automáticamente horarios para el conjunto de asignaturas específicas de cada usuario.

En base a estas circunstancias se ha determinado utilizar un sistema de base de datos relacional. Tras considerar nuestras posibilidades, se escoge ***MySQL*** [7]. Se trata de un sistema de base de datos relacional basado en el lenguaje ***SQL*** (*Structured Query Language*).

Elegimos esta opción por varias razones:

- *MySQL* es uno de los sistemas de bases de datos más extendidos. Esto facilita encontrar recursos para el mismo, incluyendo conectores con Python, que serán necesarios para la interacción entre la base de datos y el resto de la aplicación.
- El lenguaje *SQL* es relativamente sencillo y bastante legible.
- Es multiplataforma.
- Adaptarse al sistema no supondrá un gran esfuerzo a posibles programadores o administradores de la base de datos en el futuro.
- Sin dejar de ser legible y accesible, *MySQL* constituye una herramienta versátil, robusta y fiable.
- Permite gestión de usuarios y privilegios, importante para el futuro de la base de datos.

Una vez construida la base de datos, será necesario introducir datos desde distintas fuentes.

Se elaborarán distintos programas en *Python*, el lenguaje de programación utilizado en la aplicación original; este lenguaje fue creado con la legibilidad de código como prioridad principal. Los programas de introducción de datos son generalmente independientes de la aplicación de horarios, y se utilizan para introducir información aplicable a un periodo concreto. A pesar de esta independencia, se elaborarán en Python para mantener mayor homogeneidad en el proyecto.

Los distintos programas independientes y módulos a integrar en la aplicación deberán interactuar con la base de datos mediante un conector de *Python*.

Finalmente, durante el desarrollo del proyecto surgirá la necesidad de operar con documentos de *Microsoft Excel*, en formato *.xlsx*. Con este fin se utilizará la librería de *OpenPyXL* [6].

La construcción de un sistema de almacenamiento efectivo, más convencional que un archivo *XHSTT* ha sido el principal objetivo del proyecto. De su desarrollo han surgido posteriormente nuevas necesidades y posibilidades que se han convertido a su vez en nuevos objetivos. Es por esa razón que este proyecto no ha seguido el flujo de desarrollo de un proyecto software tradicional.

En sí la totalidad del proyecto constituye un módulo a integrar en el sistema preexistente, y a su vez a base de datos y los distintos programas que interactúan con ella forman una subestructura que a su vez es modular. Cada una de esas partes ha sido diseñada, desarrollada y probada, individualmente y como parte del sistema; sin embargo, los distintos módulos se ha planteado a medida que se han demostrado o implantado los anteriores, sirviendo de respuesta a una necesidad o propuesta de funcionalidad; cada bloque ha supuesto un ciclo de desarrollo relativamente independiente del resto.

El flujo de desarrollo global del proyecto se aproxima a los paradigmas *ágil* e *incremental*.

3 Diseño y construcción de la Base de Datos

La base de datos se ha construido sobre *MySQL* [7], un sistema de administración de Base de Datos Relacional de código abierto.

La estructura y terminología de las tablas y campos de la base de datos están basadas en las que utiliza *XHSTT* [2], utilizado en el proyecto original en el que se proyectó la aplicación [Vejo Gutiérrez, 2017]. Se trata de un formato de representación de datos basado en *XML* para problemas de horarios escolares. Una vez se construye un modelo con toda la información necesaria se pueden aplicar algoritmos que generen una solución que respete las restricciones definidas, también en formato *XHSTT*. En el caso del proyecto anterior, se utilizó para ello la librería *KHE* [3], de código abierto en *ANSI C*.

Diseñar de este modo la base de datos determina que, si en el futuro surge el interés o la necesidad, sea posible construir fácilmente un programa de conversión del contenido de la base de datos hacia *XHSTT*.

3.1. Tablas de la Base de Datos

Estas son las tablas generadas de la síntesis de a estructura de *XHSTT*. Se ordenan comenzando por las más básicas, seguidas de aquellas que las referencian, siendo dependientes de las primeras.

days

Tabla sencilla que enumera los días de la semana. Esta tabla existe únicamente para ser referenciada por la tabla *times*.

Podría plantearse un modelo de base de datos en los que datos como estos existiesen únicamente como enumerados en las tablas que los referencian. Optamos por dedicarles una tabla propia para posibilitar futuras modificaciones sin alterar las propiedades de una tabla y sus columnas. Si existe esta tabla dedicada los cambios podrán realizarse simplemente insertando o borrando una entrada.

Estructura de ***days***		
Columna	*Descripción*	*Referencia (FK)*
id	Identificador único del día; PK	-
name	Nombre informal	-

timegroups

Tabla sencilla, auxiliar a *times*, que incluye las categorías entre las que se puede clasificar un bloque de tiempo, funcionando de forma transversal con el día.

Estructura de ***timegroups***		
Columna	*Descripción*	*Referencia (FK)*
id	Identificador único de la categoría de periodo; PK	-
name	Nombre informal	-

Actualmente esta tabla contiene únicamente tres entradas; se contemplan los periodos por la mañana antes del descanso, por la mañana tras el descanso y por la tarde.

times

Tabla que recoge cada uno de los bloques de tiempo en los que se divide la semana.

Estructura de ***times***		
Columna	*Descripción*	*Referencia (FK)*
id	Identificador único del bloque de tiempo; PK	-
name	Nombre informal	-
day	Día de la semana	days(id)
timegroup	Categoría del periodo	timegroups(id)

En la versión actual de la base de datos se trabaja con 50 bloques de tiempo, es decir, 10 por cada día de la semana. La siguiente tabla sirve de ejemplo de la sucesión de periodos de un día cualquiera:

Ejemplo de día - Lunes		
time	*day*	*timegroup*
Lunes1	Lunes	AntesDescanso
Lunes2	Lunes	AntesDescanso
Lunes3	Lunes	DespuesDescanso
Lunes4	Lunes	DespuesDescanso
Lunes5	Lunes	DespuesDescanso
Lunes10	Lunes	DespuesDescanso
Lunes6	Lunes	Tarde
Lunes7	Lunes	Tarde
Lunes8	Lunes	Tarde
Lunes9	Lunes	Tarde

Este ejemplo ilustra la forma en que evoluciona la manera en que nuestra representación de los datos puede evolucionar cuando debe adaptarse a nuevas necesidades. Como podemos ver el bloque 10 del día aparece en la 6ª posición de la mañana. Esto se debe a que en la representación realizada en *XHSTT* sobre los datos del curso pasado sólo se contemplaban 5 horas antes de la pausa del mediodía, y hubo que añadir ese time para representar correctamente los horarios de este curso.

resourcetypes

Esta tabla contiene una de las subdivisiones en las que se clasifican los recursos contenidos en la tabla *resources*. Cada recurso se asocia a un único tipo de recursos.

Estructura de ***resourcetypes***		
Columna	*Descripción*	*Referencia (FK)*
id	Identificador único; PK	-
name	Nombre informal	-

resourcegroups

A su vez, esta tabla representa los distintos grupos en los que pueden encuadrarse cada uno de los recursos de la tabla *resources*.

Estructura de ***resourcegroups***		
Columna	*Descripción*	*Referencia (FK)*
id	Identificador único; PK	-
name	Nombre informal	-
resourcetype	Tipo de recurso	resourcetypes(id)

Estas dos últimas tablas funcionan en paralelo como clasificaciones de los recursos. Si bien en nuestro planteamiento de grupos y tipos pueden parecer informaciones redundantes, es conveniente mantenerlas por separado con el fin de asegurar la correspondencia entre esta organización de los datos y la representación equivalente en formato *XHSTT*.

En conjunto, estas tablas permiten distinguir entre profesores, grupos de alumnos (similar a los cursos) y aulas, existiendo a su vez varias subdivisiones de estas últimas (de acuerdo con el tamaño o con su clasificación como laboratorios o aulas ordinarias).

resources

Tabla que contiene las instancias de recursos con los que cuenta la universidad. Se incluyen tres principales grupos de recursos: profesores, aulas y grupos de alumnos. De nuevo, esta organización responde a la estructura de *XHSTT*.

Estructura de ***resources***		
Columna	*Descripción*	*Referencia (FK)*
id	Identificador único; PK	-
name	Nombre informal	-
resourcetype	Tipo de recurso	resourcetypes(id)
color	Exclusivo a las aulas. Color utilizado para la representación visual	-

resources_to_resourcegroups

Tabla de enlace entre *resources* y *resourcegroups*, que permite asociar cada recurso a múltiples grupos (por ejemplo, LABORATORIO_1 está asociado a los grupos gr_Room, gr_Laboratorio y gr_Aula_Normal). En otras palabras, posibilita relaciones *many to many*.

Estructura de ***resources_to_resourcegroups***		
Columna	*Descripción*	*Referencia (FK)*
resource	Recurso a asociar; PK	resources(id)
resourcegroup	Categoria de recursos asociada; PK	resourcegroups(id)

eventgroups

Esta tabla contiene todas las asignaturas que se imparten en la facultad.

Estructura de ***eventgroups***		
Columna	*Descripción*	*Referencia (FK)*
id	Identificador único; PK	-
name	Nombre informal	-
curso	Curso al que pertenece la asignatura	-
cuatrimestre	Cuatrimestre en el que se imparte la asignatura	-
h_teoria	Número de horas de teoría en la asignatura	-
h_practica	Horas de horas de prácticas en la asignatura	-

Para esta tabla, se utiliza como *id* el código de la asignatura de la universidad (por ejemplo, G1176 para *astronomy*).

Los cuatro últimos campos se han incorporado al modelo actual, pero no han llegado a utilizarse.

events

Modelo de la tabla que contiene los eventos a representar en el horario, o lo que es lo mismo, cada una de las clases que deben impartirse para cada asignatura a lo largo de una semana.

Estructura de ***events***		
Columna	*Descripción*	*Referencia (FK)*
id	Identificador único; PK	-
name	Nombre informal	-
eventgroup	Código de la asignatura a la que corresponde este evento (clase)	eventgroups(id)
time	Bloque de tiempo de la semana que ocupa la asignatura	times(id)

events_need_resources

Tabla de enlace entre *events* y *resources*, que permite enlazar los recursos necesarios a cada evento. Permite generar relaciones *many to many*.

Estructura de ***events_need_resources***		
Columna	*Descripción*	*Referencia (FK)*
event	Evento al que asociar recursos; PK	events(id)
resource	Recurso a asociar; PK	resources(id)

Estas dos últimas tablas se han definido como modelos con la idea de que cada cuatrimestre puedan generarse instancias particulares en las que incluir la configuración adecuada de eventos. Esto permite copiar la estructura para cada periodo y elaborar un horario que se ajuste a las necesidades específicas que hayan surgido ese año.

Por ejemplo, durante este proyecto se han generado las tablas *events_cuatrimestre2_1717* y *events_need_resources_c2_1718* para demostrar y probar las funcionalidades de la base de datos con datos reales del segundo cuatrimestre del curso 2017-2018.

3.2. Impacto de la Base de Datos

La estructura con la que se ha diseñado e implementado la base de datos asegura de forma innata cierta medida de coherencia en los datos que no estaba presente en la representación en *XHSTT*.

En primer lugar, las claves primarias (*primary keys*) de las tablas aseguran la ausencia de entradas duplicadas, redundantes y potencialmente inconsistentes; en ningún caso el sistema nos permitirá introducir datos que vulneren estas reglas básicas. Dependiendo de la instrucción utilizada (*INSERT* o *INSERT IGNORE*), el sistema detendrá la ejecución de nuestras queries o ignorará instrucciones individuales. En ningún caso se permitirá una introducción o modificación que genere una incidencia de este tipo.

En segundo lugar, las claves ajenas (*foreign keys*) generan una estructura interconectada, que extiende la coherencia a las relaciones entre tablas. Para poder insertar o modificar cada entrada de una tabla, deberá mantenerse la consistencia en sus relaciones con todas las tablas a las que referencia.

4 Proceso de introducción de datos

Una vez contamos con una base de datos funcional, con una estructura apropia para nuestros objetivos, es el momento de introducir los datos.

En este caso hemos utilizado datos reales, con una doble función:

- Comprobar que el funcionamiento de cada parte del sistema, individualmente y en conjunto, es correcto.
- Servir de base funcional del sistema, de modo que pueda ser utilizado pronto, incluso mientras dure su desarrollo.

La fuente de la mayoría de los datos relevantes son archivos en formato *.xlsx*. Para acceder a ellos se hará uso de la librería de *Python OpenPyXL* [6]. Esta permite leer, crear y modificar archivos y hojas de cálculo.

El objetivo en esta sección será la introducción de los datos relevantes de la forma más sencilla y eficiente. Si es posible, se diseñarán códigos que iteren sobre los archivos de entrada sin realizar modificaciones sobre el mismo.

Seguiremos la misma estructura del capítulo anterior para explicar los medios utilizados o diseñados para introducir los datos.

days Esta tabla contiene un número muy limitado de datos (actualmente *'Lunes'*, *'Martes'*, *'Miercoles'*, *'Jueves'* y *'Viernes'*). Se introducen directamente mediante *query* sobre la base de datos.

timegroups De nuevo, se trata de un rango de valores muy limitado. Se podría expresar como enumerado en el campo de la tabla que lo referencia, pero eso complicaría cualquier alteración futura que el sistema precisase. Como en el caso anterior, al tratarse de sólo 3 valores (*'AntesDescanso'*, *'DespuesDescanso'* y *'Tarde'*) en esta versión, se introducen directamente.

times En este caso ya se puede iterar sobre valores enumerados (días de la semana), aunque para cada uno deben introducirse 10 entradas de forma explícita, por tratarse de combinaciones de valores específicas.

Para ello, hemos decidido automatizar este proyecto, programando un pequeño *script* en *Python* (*FillTimes.py*) que genera diez horas hábiles para cada día. Después de varias reuniones se fijo diez horas lectivas por día, aunque este número pudiera cambiar en el futuro.

resourcetypes Contiene un conjunto de valores limitados. Por tanto, se introducen directamente mediante *query* en el gestor de base de datos.

resourcegroups Como en el caso anterior, esta tabla cubre un conjunto ve valores específicos limitados e independientes de otras tablas. Se introducen por tanto directamente mediante *query*.

resources* y *resources_to_resourcegroups Estas dos tablas contienen los distintos recursos que maneja el sistema y los grupos a los que pertenecen. Por lo tanto, ambas se rellenan en paralelo.

Sin embargo, ya que estas tablas tratan tres tipos de recursos distintos, será necesario enfocar el tratamiento e introducción de cada tipo de datos de forma independiente.

- ***Aulas (Rooms):*** En este caso nos encontramos con un conjunto de elementos muy limitado que, si bien es más numeroso que los grupos de tiempo, los días y similares, sigue siendo específico. Ya que no existe una definición previa de cada elemento, estos son definidos individualmente (en base a la información recogida en el proyecto predecesor , que clasificaba las clases de acuerdo con un criterio de tamaño, y basándonos en un archivo de horario para asignar a cada aula un color) e introducidos mediante *query* directamente sobre la base de datos.

- ***Grupos de alumnos (Classes):*** Esta tabla ha pasado por varias versiones, a medida que iba evolucionando y probándose el proyecto.

  En una primera versión, los grupos de alumnos se tomaron directamente de los archivos en *XHSTT* del proyecto anterior, que definía varios grupos para cada curso. Cada tupla de datos se introdujo mediante *queries* directas sobre cada tabla (*resources* y *resources_to_resourcegroups*).

  Sin embargo, al considerar la estructura de grupos definida, una división uniforme por curso, en comparación con la organización real, una división específica a cada asignatura (y en ocasiones diferente para horas de problemas y laboratorios), realizada por cada departamento de forma independiente; las distintas divisiones comúnmente cambian cada curso. Ya que no existe un registro centralizado que permita sintetizar de forma efectiva esta información, se concluye operar con un único grupo por curso. Estas modificaciones se realizan mediante query directa.

  Finalmente, se decide recuperar la división en grupos interna al curso para las menciones y diplomas especiales de cuarto curso. Esta división se define mediante *query* sobre la base de datos.

- ***Profesores (Teachers):*** Para llenar esta tabla hemos utilizado un registro de profesores junto a las asignaturas que enseñan facilitado por la Universidad. La lista tiene una estructura clara y regular, lo que permite iterar sobre ella. Se divide en dos archivos: uno para los grados y otro para el doble grado, pero esto no supone un gran problema.

  Para procesar estos datos se construye el programa *LeeProfesores_20172018.py*. Este procesa las filas del archivo *.xlsx* especificado como entrada, extrayendo las columnas referentes a la información del profesor e ignorando las de la asignatura en cada fila. Ya que puede repetirse profesor para varias asignaturas, nos aseguramos de que el proceso no se detenga al encontrar duplicados utilizando una instrucción *INSERT IGNORE*, que simplemente descarta las entradas duplicadas, sin detener la ejecución.

  Para lanzar el programa aplicado a distintos archivos simplemente se requiere modificar la ruta del archivo *.xlsx* correspondiente y los límites de las filas útiles de la lista (las que aparecen entre el primer y el último profesor). Esto permitirá su adaptación a futuras listas de forma sencilla.

eventgroups Las asignaturas se extraen del mismo archivo que los profesores, seleccionando distintas columnas.

El *script* de *Python LeeAsignaturas.py* realiza esta función, iterando sobre el mismo archivo que el script de profesores pero centrándose en distintos campos.

events_cuatrimestre2_1717* y *events_need_resources_c2_1718 Se trata de las tablas instanciadas de *events* y *events_need_resources* para los datos del segundo cuatrimestre de 2018. Estas tablas constituyen la parte esencial de la base de datos con las que interactúa la aplicación. en su construcción se referencian datos de todas las tablas anteriores; como es natural, el proceso de introducción de datos en estas tablas es el más complejo de todos.

Ya que no existe una fuente de información que contenga los datos relevantes a estas tablas estructurados de forma regular y tratable por un programa iterativo, es necesario crearlo. Para esta parte del proceso de introducción de datos, es necesario elaborar de forma manual un archivo que sintetice toda la información relevante. Es un proceso poco eficiente, pero actualmente no existe una alternativa.

De este modo se realiza la conversión manual de los horarios existentes para el cuatrimestre a un archivo de Excel, que al completarse contiene la lista de eventos semanales de todas las asignaturas junto con los datos que no pueden obtenerse de ninguna otra fuente.

Para procesarlo todo se crea el script *LeeHorarioTratado.py* que procese todos los datos de entrada. En él, se itera sobre la lista contenida en el archivo generado. Para cada entrada se extrae el *id* del evento, el de la asignatura al que corresponde, el aula que requiere, el bloque de tiempo que ocupa y los cursos a los que se imparte. Cada fragmento de información recibe un tratamiento distinto dependiendo del tipo de dato involucrado:

- *Eventos en* ***events_cuatrimestre2_1718****:* Por cada entrada extraída de la lista manufacturada se introduce el evento correspondiente en la base de datos; esto incluye el código de evento, el de la asignatura, y el bloque de tiempo.

- *Cursos en* ***events_need_resources_c2_1718****:* Cada entrada de la lista incluye campos que representan los cursos en los que cada clase se imparte en cada grado (si no se imparte estará en blanco). Se procesa cada entrada y, una vez introducido el evento, se introduce la asociación a los grupos (tipo *Class*) de alumnos correspondientes.

- *Aulas en* ***events_need_resources_c2_1718****:* De nuevo, cada entrada de la lista incluye el aula que el evento utiliza. Una vez se ha introducido el evento, se lee e introduce el aula correspondiente.

- *Profesores en* ***events_need_resources_c2_1718****:* Representan los profesores que requerirá cada evento de clase; son la categoría de datos más fácil de procesar en las dos últimas tablas. Las asociaciones profesor-asignatura se obtienen de la lista de la que se extrajeron anteriormente esas dos categorías de dato por separado.

  La asignación de profesores a eventos de clase se realizará a medida que estos se introducen en la primera tabla.

De este modo habríamos introducido un cuatrimestre completo en la base de datos. Sin embargo, este procedimiento plantea un gran problema en cuanto a la visión de futuro del proyecto.

Si bien la mayoría de los datos que cambien podrán introducirse simplemente modificando el archivo fuente y los límites sobre los que operar, para las tablas de eventos y la de asignación de recursos no existe una convención en cuanto a formato de entrada (por eso ha sido necesario elaborar una lista propia). Si se pretende utilizar este sistema en posteriores años lectivos, esta debería ser una de las principales prioridades.

5 Interacción con la aplicación

En este punto contamos con una Base de Datos que funciona correctamente y que contiene los datos necesarios para representar un problema de gestión de horarios escolares, incluyendo la solución.

Es el momento de integrar este nuevo módulo en la aplicación que se generó en el proyecto original [Vejo Gutiérrez, 2017]. La base de datos sustituirá a los archivos *.xml* en formato *XHSTT* [2] como sistema de almacenamiento de información.

En la versión de la aplicación generada con este proyecto no se utilizarán esos archivos. Sin embargo, un desarrollo futuro podría generar un sistema de conversión que tradujese nuestra base de datos en un archivo en *XHSTT*. Si bien este no será útil como sistema de almacenamiento (la base de datos realiza esa función, dejándolo obsoleto en ese aspecto), siempre podría utilizarse como formato de representación del problema, al que después aplicarle algún software de generación de soluciones, como *KHE* [3].

5.1. Ejemplos de *Queries*

En lo que a código *Python* se refiere, en este proyecto he estado encargado de aquellos módulos que interactúan únicamente con la base de datos. Esto incluye todos los sistemas de entrada de datos descritos en el capítulo anterior y un programa para exportar horarios a documento de *Excel* que se explicará en el siguiente.

Por el contrario, en el proceso de integración de la nueva base de datos con la aplicación preexistente, mi tarea ha sido únicamente generar las *queries SQL* necesarias para cumplir con los casos de uso definidos en el proyecto original.

Ya que muchos de los casos de uso requieren instrucciones muy similares entre ellos, se hará una selección de *queries* representativas del trabajo realizado. Usaremos los que representan las funcionalidades originales de la aplicación, que requieren los principales tipos de operaciones utilizados.

Cargar datos de horarios: Para consultar alguno de los horarios de la aplicación, el usuario selecciona un filtro y pulsa el botón de cargar. Esto supone dos operaciones sobre la base de datos cada cual con su *query*:

1. En primer lugar, es necesario cargar los valores posibles por los que filtrar resultados. La aplicación presenta varios menús emergentes, cada uno representando una categoría de filtrado (aula, profesor, grado, curso...). Cuando el usuario pulsa sobre uno de los botones que muestran uno de los menús, se lanza una query que extrae todos los elementos de ese grupo.

   Por ejemplo, para cargar el filtro por aulas:

   ```
   SELECT * FROM resources WHERE resourcetype='Room'
   ```

2. En segundo lugar, una vez el usuario selecciona un valor del filtro, se lanza una segunda *query* sobre la base de datos que devuelva los eventos correspondientes al valor elegido.

   Siguiendo con el ejemplo anterior:

```
SELECT * FROM events_cuatrimestre2_1718
INNER JOIN events_need_resources_c2_1718 enr ON
events_cuatrimestre2_1718.id=enr.event
INNER JOIN resources res ON enr.resource=res.id
INNER JOIN resources resRef ON enr.resource=resRef.id
WHERE resRef.id= [AULA SELECCIONADA]
```

Esto devolverá todos los eventos que tienen asignada el aula elegida junto a todos los demás recursos asociados. No es la única implementación posible (puede que el resto de los recursos no nos interesen, por ejemplo).

Una vez ejecutado esto, el programa tratará la tabla resultante y mostrará la información correspondiente al usuario.

Intercambio de clases: El usuario selecciona dos eventos de clase cargados e intercambia los bloques de tiempo (periodos del horario) en los que se imparten.

Asumimos que se ha cargado el horario filtrado como se explicó anteriormente. Ahora el usuario selecciona la opción intercambiar y elige dos clases. Entonces se ejecuta una query para cada uno de los eventos:

```
UPDATE events_cuatrimestre2_1718
SET time=[TIEMPO DE LA OTRA CLASE]
WHERE id=[ID DEL EVENTO QUE SE ESTÉ TRATANDO]
```

Modificar aula: El usuario selecciona un evento, pulsa el botón de cambiar aula y elige del menú emergente la nueva aula, que sustituirá a la anterior.

Esto implica lanzar las dos *queries* descritas antes para cargar filtros y buscar elementos filtrados. Además, a estas se suma una nueva *query* que modificará el recurso asociado al evento.

```
UPDATE events_need_resources_c2_1718
SET resource=[ID DEL AULA NUEVA]
WHERE event=[ID DEL EVENTO QUE SE ESTÉ TRATANDO]
AND resource=[ID DEL AULA ANTIGUA]
```

Mostrar incidencias: El usuario pulsa un botón y el sistema le muestra las colisiones que se producen en el uso de recursos de acuerdo a la configuración actual del problema.

Eso se realiza lanzando una *query* sobre la base de datos que muestra todos los casos en los que un recurso está siendo utilizado por dos eventos distintos en un mismo bloque de tiempo.

A continuación se muestra una posible implementación de la *query*.

```
SELECT resources.id, resources.name, ecA.name as 'Evento A', ecB.name as 'Evento B',
ecA.time FROM resources
INNER JOIN events_need_resources_c2_1718 enrA ON resources.id=enrA.resource
INNER JOIN events_cuatrimestre2_1718 ecA ON ecA.id=enrA.event
INNER JOIN events_need_resources_c2_1718 enrB ON resources.id=enrB.resource
INNER JOIN events_cuatrimestre2_1718 ecB ON enrB.event=ecB.id
WHERE ecA.id!=ecB.id AND ecA.time=ecB.time
¿AND resources.resourcetype=[TIPO]?
```

La línea contenida entre interrogaciones al final de la query, parte de la cláusula *WHERE*, haría posible filtrar colisiones de eventos de un sólo tipo.

El resto de *queries* que se usan en la aplicación son adaptaciones o versiones muy similares de las aquí descritas.

6 Implantación y nuevas funcionalidades

Tras terminar de construir y rellenar la base de datos, se instala la aplicación en el equipo de la jefa de estudios. Hasta el momento se trata de una instalación local de la aplicación con el nuevo módulo integrado.

Esto precede a algunas de las funcionalidades explicadas en el capítulo anterior, que se han incorporado al sistema de forma progresiva. Se toma la decisión de poner el sistema parcialmente funcional en manos de usuarios finales para agilizar el proceso de pruebas por parte de los *stakeholders*. Aunque se han realizado pruebas internas a medida que se desarrollaba el proyecto y se integraban sus componentes, estas serán las que demuestren si todo funciona correctamente o aparecen problemas derivados de un uso real y continuo de la aplicación.

Como consecuencia, esas pruebas se realizan en paralelo con el desarrollo de las funcionalidades posteriores. Estas cubren desde nuevas opciones menores de la aplicación, resueltas mediante queries sencillas, hasta módulos complejos totalmente independientes de la misma (Por ejemplo, el conversor a *Excel*). A medida que se completaban, estos módulos eran integrados en el sistema final para ser probados al darles uso.

A medida que se iban completando las funcionalidades básicas del sistema se proponían también posibles cambios y nuevos casos de uso a añadir al proyecto. En el siguiente apartado se describen algunos.

6.1. Problemas, soluciones y nuevas funcionalidades

Una vez se ha comenzado a dar un uso normal a la aplicación, se ha podido observar fallos y carencias a reparar. A continuación se listarán las categorías más representativas:

- **Dato erróneo:** El tipo de fallo más común. No se deben a errores en la construcción de la base de datos, sino a la introducción de algún valor incorrecto en los archivos desde los que se leyeron los datos a incorporar a la base de datos.

  Este tipo de errores se solucionan generalmente modificando la información desde la aplicación si es posible. En los pocos casos encontrados durante las pruebas donde no ha sido posible, se soluciona lanzando una *query* específica sobre la base de datos.

  No es un problema demasiado grave, principalmente porque no supone un error en el sistema y tiene fácil solución. Dado que se ha comenzado a trabajar con el sistema con datos de prueba que han debido introducirse a mano, es normal que aparezcan algunos errores.

  No suponen una gran amenaza a largo plazo. Ya que cada cuatrimestre se plantea un problema independiente, no se iterará sobre los datos incorrectos de periodos anteriores. Además, si se acuerda un formato estándar de entrada de datos de horario, este problema podría desaparecer casi por completo.

- **Funcionalidad necesaria:** Durante el desarrollo del proyecto, el uso común de la aplicación por sus usuarios finales ha demostrado necesidades en lo que a funcionalidad se refiere. Esta clase de problema es más complicado, ya que atañe a la estructura interna del modelo de representación de conocimiento.

En ocasiones aparece una necesidad de nuevas opciones en la aplicación. Un ejemplo sería la inclusión de la opción que permite visualizar todas las aulas libres a una hora concreta. Algo así se resuelve con una o unas pocas *queries* sencillas.

En otros casos no es tan sencillo, ya que las carencias pueden responder a un planteamiento del sistema con defectos. Estas necesidades no siempre pueden suplirse, bien porque falten los datos relevantes o porque no sea realista contemplarlos. Por ejemplo, la cuestión de los grupos de alumnos, ya descrita en el capítulo 3 (sección de la tabla resources con el valor de resourcetype 'Class').

Si consideramos la representación de esta información en una escala que va de lo más complejo a lo más simple, veríamos que la realidad de la situación está en el extremo más complicado: cada asignatura tiene una división en grupos independiente del resto. Es más, en algunos casos hay una división diferente para las horas de *problemas en aula* y *prácticas de laboratorio*. Estas divisiones son realizadas por los profesores que la imparten, independientemente de la jefatura de estudios, para la que la estructura es invisible.

Podríamos ser más granulares incluso y reconocer que cada alumno, matriculado de un conjunto de asignaturas distinto, pertenecerá a un subconjunto de grupos que a menudo será único.

Cualquiera de estos dos enfoques es inviable para nosotros, no tanto porque no podamos concebir una estructura de base de datos que represente la situación al completo, sino porque la realidad limita nuestras posibilidades: en el caso de los grupos de cada asignatura, no existe un registro al que pueda accederse para utilizarlo como fuente de información; en el caso de los grupos por alumno, a ese primer problema se le suma el crecimiento exponencial de la complejidad del problema.

Cerca del extremo más simple de la escala se encuentra la solución implementada originalmente. Se contempla únicamente un grupo por grado y curso. Esto supone una visión del sistema efectiva pero demasiado sencilla. Las carencias que se observan tras la implantación del sistema lo demuestran.

Por lo tanto, se acuerda un término medio: se contemplará un grupo para los tres primeros cursos, más uno por mención y/o diploma especial para el cuarto curso. Esta solución se integra en la aplicación ejecutando queries especiales sobre la base de datos.

Estos han sido los tipos de problemas principales que se han encontrado. La estructura de la base de datos en sí no ha presentado errores propios.

Aunque no responde a un problema existente, la principal funcionalidad nueva en la que se ha trabajado es el conversor a *.xlsx* que se describe en el apartado siguiente.

6.1.1. Conversor a *Excel*

Una de las nuevas funcionalidades desarrolladas para la aplicación ha sido un mecanismo que exportase horarios a un archivo de *Microsoft Excel* (formato *.xlsx*). Esta fue una de las ideas propuestas al principio del proyecto que pasó a ser una prioridad y comenzó a desarrollarse una vez completada la construcción de la base de datos.

El nuevo módulo constituye un programa independiente del resto de la aplicación, que interactúa únicamente con la base de datos.

A continuación se describen las distintas fases del proceso de exportación de datos a archivo de *Excel* y cómo se han implementado. Para realizar el proceso, se necesitan definir una serie de parámetros: credenciales de acceso a la base de datos, ruta de creación del archivo en el equipo y composición del archivo (hojas que lo componen y grupos de alumnos representados en cada una). A grandes rasgos, el proceso es el siguiente:

1. **Conexión a la base de datos:** El programa comienza como cualquier otro, conectándose a la base de datos con los credenciales correspondientes.

2. **Creación del archivo:** Haciendo uso de *Openpyxl* [6], se crea el archivo en la ruta especificada.

3. **Creación de una estructura intermedia auxiliar:** Para contener todos los datos de forma organizada a medida que van extrayéndose se crea una estructura auxilia, un diccionario de *Python* en que cada *key* represente un bloque de tiempo y cada valor una lista de eventos asociados.

4. **Extracción de los datos:** De acuerdo a los parámetros de entrada, se van creando *worksheets* en el archivo *.xlsx*. Para cada uno de los grupos asociados, se lanza una query que extrae todos los eventos que referencian a esos grupos, enlazados a su vez con su aula y la información de la asignatura correspondiente.

   Se extraen los datos relevantes y se almacenan en la estructura intermedia.

5. **Inserción de datos en el archivo:** Una vez se han extraído y estructurado todos los datos pertinentes a una hoja de *Excel*, el conjunto se inserta en el archivo.

   Mediante la librería *Openpyxl*, se trasladan los datos del diccionario auxiliar a la hoja de cálculo (previamente maquetada).

Los pasos 4 y 5 se repiten para cada hoja definida para el archivo de destino, vaciando cada vez la estructura intermedia auxiliar.

Como resultado, tenemos un sistema que produce de forma efectiva representaciones visuales de horarios de clase. La estructura de estos es similar a la de los horarios creados a mano: cada casilla contiene la información de los eventos que tienen lugar en el bloque de tiempo correspondiente. Para facilitar la visualización, se colorean las casillas de acuerdo con el aula en que el evento tiene lugar.

En una primera versión del programa, este estaba limitado a la asignación de un grupo por hoja del archivo. Posteriormente se ha modificado el código para permitir definir múltiples grupos en una misma hoja.

Esto permite, por ejemplo, representar juntas las asignaturas consideradas comunes o troncales junto a las de mención de un curso concreto. Pensando en el futuro, este sistema permitirá crear horarios personalizados para usuarios, definiendo un conjunto de grupos que correspondan a su curso o asignaturas.

7 Conclusiones y futuro del proyecto

Es el momento de evaluar lo que se ha alcanzado, comparándolo con los objetivos iniciales y determinando en qué puntos se ha tenido éxito y cuáles se han dejado como trabajo futuro.

7.1. Conclusiones

En general se ha cumplido con todos los objetivos propuestos al comienzo del desarrollo del proyecto.

El objetivo principal, la creación de una base de datos que sustituyese al archivo *XHSTT* [2] como capa de datos de la aplicación se ha completado con éxito. En su estado resultante, el sistema interactúa con el nuevo módulo de forma invisible para los usuarios, cumpliendo con las funcionalidades necesarias.

Además, como objetivos secundarios, se ha logrado dotar a la aplicación de nuevas funcionalidades (incluyendo opciones básicas y soluciones avanzadas como el conversor *.xslx*).

Como contrapartida, han existido aspectos en los que me hubiera gustado lograr algo más o completarlo más rápido. Por ejemplo, el proceso de introducción de datos resultó más complicado y laborioso, y consumió mucho más tiempo de lo que había estimado al principio del desarrollo. La fase se completó correctamente, pero la exigencia de tiempo y esfuerzo que supuso deberán ser consideradas en relación con el futuro de la aplicación.

Finalmente, como conclusión personal, cabe comentar que este trabajo me ha servido como aprendizaje y fuente de experiencia. Ha constituido una labor exigente, requiriendo tiempo, esfuerzo y compromiso. En algunos frentes se han conseguido logros, mientras que en otros he debido comprometerme, aceptando mis limitaciones y circunstancias.

Globalmente ha supuesto una experiencia satisfactoria y valiosa, y ha reportado un resultado del que estoy orgulloso.

7.2. Futuro

Del mismo modo que este proyecto surge como continuación de un Trabajo de Fin de Grado realizado anteriormente por otro alumno, cabe esperar que haya espacio para realizar mejoras y cambios.

En un proyecto como este, que pretende poner solución e implantar mejoras en un sistema complejo y que lleva tiempo en uso, es complicado predecir cuánto trabajo resta, más allá de que es mucho. Es difícil estimar todas las posibilidades que existen para con el futuro de nuestro sistema, y es difícil estimar el tiempo que supone el desarrollo e implantación de nuevas funcionalidades, ya que las necesidades cambian con el tiempo, y el sistema nunca deja de estar en uso.

Desde el punto en el que nos encontramos, al finalizar el desarrollo de este TFG, algunos de los objetivos o posibles nuevos planteamientos serían los siguientes:

- **Fijar un estándar para la entrada de los datos:** Se trata del objetivo más prioritario. Este trabajo ha requerido generar archivos que sintetizasen los horarios del cuatrimestre de forma legible y regular, con el fin de que el programa pudiese iterar de forma efectiva sobre ellos.

Tanto si se adopta un formato similar al utilizado de forma provisional en este proyecto como otro totalmente nuevo, adaptar los módulos de lectura no debería suponer una tarea demasiado complicada.

- **Evaluar la situación de los grupos:** Durante el desarrollo del trabajo hemos pasado por distintas versiones en cuanto a la representación de grupos de alumnos se refiere. Pensando en años posteriores, podría ser conveniente plantearse si el enfoque de nuestra base de conocimiento es correcto o habría que cambiar algo. Debería tenerse en cuenta la manera en la que actualmente se tratan los grupos en la Universidad.

- **Hacer remoto el sistema:** Es una posibilidad que merece ser considerada. Esto supondría alojar el sistema en un servidor. Algo así podría permitir el acceso a la aplicación desde cualquier lugar.

  Esto sería instrumental si se decide distribuir la labor de creación y gestión de horarios.

- **Servicio de horarios personalizados:** En relación con lo anterior, supondría generar un servicio que permitiese generar horarios semanales personalizados para distintos usuarios. La versión final del conversor a *Excel* está diseñada de forma que puedan concatenarse grupos a representar en una misma hoja de cálculo.

  Como alternativa, también sería posible modificar mínimamente ese módulo para que funcionase en base a conjuntos de asignaturas personalizados en lugar de grupos (varias tuplas de curso y asignatura para cada alumno).

A Instrucciones de uso de la aplicación

La aplicación fue elaborada como parte del primer proyecto [Vejo Gutiérrez, 2017]. Yo no he trabajado en su interfaz gráfica ni la mayoría de su código. Sin embargo, generar *queries* en *SQL* y definir métodos de extracción, introducción y modificación de datos a un nivel abstracto han constituido algunas de mis responsabilidades.

Este apartado pretende servir de manual de instrucciones básico para operar la aplicación.

Requisitos previos:

Para poder ejecutar la aplicación necesitamos realizar una instalación relativamente compleja, que incluye los siguientes elementos:

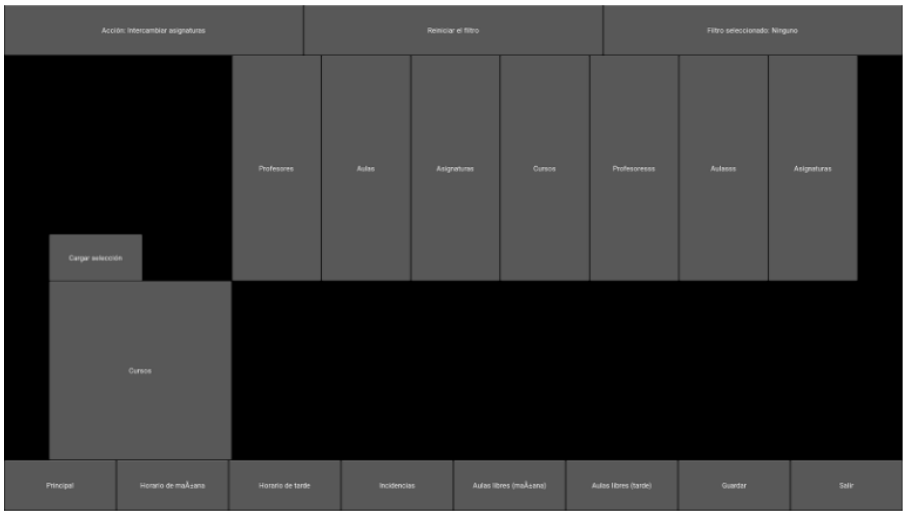
- Necesitamos instalar una base de datos *MySQL*, crear un usuario con todos los privilegios e importar un archivo *dump* que contenga todos los datos.

  Hay que introducir los credenciales de acceso en las secciones correspondientes de los archivos *ParserSQL_233.py* y *ParserSQLNew.py*.

- *Python 3* debe estar instalado en el equipo.

  Debemos resolver dependencias instalando una serie de librerías de *Python*, incluyendo *Kivy*, *lxml* y *pymysql*. Al ejecutar la aplicación se indicará si falta algo, lo que permitirá instalarlo con *pip install [módulo]* o una instrucción equivalente.

Partes principales:



Esta es la pantalla principal. A continuación se recoge una descripción simple de las partes visibles en la imagen superior o alcanzables mediante interacciones simples.

En primer lugar, el centro de la pantalla lo ocupa la parte variable de la interfaz, que tiene un aspecto distinto en cada sección (se explican en detalle más abajo). En la captura de ejemplo es visible la pantalla principal, que contiene las opciones de filtros.

En segundo lugar, los botones de la **barra superior**, de izquierda a derecha:

1. **Menú - Menú de acciones**: Permite elegir las acciones que se realiza con el cursor: intercambiar aulas, eliminar el aula del evento o asignar una de las aulas al evento.
2. **Botón - Reiniciar el filtro**: Permite eliminar los valores del filtro de eventos y devolverlo al estado inicial.
3. **Campo - Filtro seleccionado**: Espacio donde se marca el filtro que el usuario haya aplicado.

En tercer lugar, los botones de la **barra inferior**, de izquierda a derecha:

1. **Sección - Principal**: Cambia la sección visible a la inicial, que contiene el control de filtros y carga de datos. Es la sección visible en la captura de la página anterior.
2. **Sección - Horario de mañana**: Muestra el horario semanal de mañana, compuesto por los eventos correspondientes al filtro seleccionado.

   Está formado por una tabla con el evento correspondiente a cada bloque de tiempo en la casilla correspondiente.
3. **Sección - Horario de tarde**: Muestra el horario semanal de tarde, compuesto por los eventos correspondientes al filtro seleccionado.

   Está formado por una tabla con el evento correspondiente a cada bloque de tiempo en la casilla correspondiente.
4. **Sección - Incidencias**: Muestra una lista con todas las colisiones de uso de recursos en un mismo bloque de tiempo.
5. **Sección - Aulas libres (mañana)**: Muestra un horario de mañana en el que cada casilla contiene todos los espacios libres en el bloque de tiempo correspondiente.
6. **Sección - Aulas libres (tarde)**: Muestra un horario de tarde en el que cada casilla contiene todos los espacios libres en el bloque de tiempo correspondiente.
7. **Botón - Guardar**: Almacena los cambios realizados a l horario en la base de datos.
8. **Botón - Salir**: Sale del programa. Los cambios no se guardan si no se pulsa el botón anterior.

Operaciones:

Estas son las principales operaciones que permite realizar la aplicación:

Seleccionar Filtro y Cargar Eventos: En la sección *Principal*, debemos pulsar la categoría deseada y, en el menú emergente, el valor correspondiente de acuerdo al que se restringirá la información a mostrar. Finalmente, hay que pulsar el botón *Cargar selección*.

Eliminar filtro: Se realiza pulsando el botón central de la barra superior.

Visualizar horarios: Una vez cargados los datos, simplemente debemos acceder a las secciones *Horario de mañana* y *Horario de tarde*, que contienen la información extraída en forma de visualización propiamente estructurada.

Intercambiar eventos: Una vez seleccionada la herramienta correspondiente en el *menú de acciones*, basta con pulsar cualquier par de casillas de un horario para intercambiarlas.

Quitar aula: Hay que seleccionar la herramienta correspondiente en el *menú de acciones*. Si después se pulsa alguna casilla de un horario se eliminará el aula asociada al evento allí contenido.

Asignar aula: Hay que seleccionar la herramienta correspondiente en el *menú de acciones*. Después, pulsar un evento en algún horario le asignará el aula elegida.

Visualizar incidencias: Se realiza accediendo a la lista contenida en la sección *Incidencias*.

Visualizar espacios libres: Las aulas, laboratorios y seminarios libres se representan en las secciones *Aulas libres (mañana)* y *Aulas libres (tarde)*.

Guardar cambios: Para actualizar la base de datos con la información modificada en la aplicación, es necesario pulsar el botón *Guardar* de la barra inferior.

Salir de la aplicación: Para salir de la aplicación, pulsar el botón *Salir* de la esquina inferior derecha.

Referencias

Python Software Foundation website. https://www.python.org/psf-landing/.

XHSTT home page. https://www.utwente.nl/en/eemcs/dmmp/hstt/. Accessed: 2017-10-27.

Home page of KHE, howpublished = http://www.it.usyd.edu.au/~jeff/khe/, note = Accessed: 2017-11-26.

Home page of HSEval, howpublished = http://www.it.usyd.edu.au/~jeff/cgi-bin/hseval.cgi, note = Accessed: 2017-11-26.

Kivy website. https://kivy.org/. Accessed: 2017-11-25.

OpenPyXL documentation. https://openpyxl.readthedocs.io/en/stable/. Accessed: 2018-03-10.

MySQL documentation. https://dev.mysql.com/doc/.

J. R. Vejo Gutiérrez. Desarrollo de una aplicación para la planificación de horarios, 9 2017. http://hdl.handle.net/10902/12585