UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

UNIFICACIÓN, MANTENIMIENTO CORRECTIVO Y PERFECTIVO DE LOS SISTEMAS DE EVALUACIÓN ACADÉMICA DE LA FACULTAD DE MÚSICA.

R E P O R T E D E T R A B A J O
P R O F E S I O N A L

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A:

LUIS GARCÍA ORTIZ

DIRECTORA DE TRABAJO:
DRA. AMPARO LÓPEZ GAONA
2016

Datos del Jurado

1. Datos del alumno
   García
   Ortiz
   Luis
   56 60 82 50
   Universidad Nacional Autónoma de México
   Facultad de Ciencias
   Ciencias de la Computación
   306229976

2. Datos del tutor
   Dra.
   Amparo
   López
   Gaona

3. Datos del sinodal 1
   Dr.
   José de Jesús
   Galaviz
   Casas

4. Datos del sinodal 2
   M. en I.
   Gerardo
   Avilés
   Rosas

5. Datos del sinodal 3
   Dra.
   Amparo
   López
   Gaona

6. Datos del sinodal 4
   M. en C.
   María Guadalupe Elena
   Ibargüengoitia
   González

7. Datos del sinodal 5
   Dra.
   María de Luz
   Gasca
   Soto

8. Datos del trabajo escrito
   Unificación, mantenimiento correctivo y perfectivo de los sistemas de evaluación académica de la Facultad de Música.
   77 p
   2016

A mi padres, María Guadalupe y Laurencio Elpidio.
A mi hermano, Armando.

Me enseñaron que para luchar,
se permite caer, llorar y gritar.

Índice General

INTRODUCCIÓN

La Secretaría de Servicios y Atención Estudiantil de la Facultad de Música, es la encargada de brindar diversos servicios escolares que están relacionados con el personal docente, los estudiantes tanto de nivel propedéutico, como de nivel licenciatura, y a los aspirantes que desean ingresar a la Institución. La Secretaría de Servicios y Atención Estudiantil comenzó a diseñar un sistema para atender, de manera oportuna, gran parte de los servicios que requerían una sistematización como: la distribución de las asignaturas-grupos impartidas por el personal académico, la distribución de horarios de las mismas, la elaboración de actas para evaluaciones, el seguimiento de concurso de selección para ingreso tanto a nivel licenciatura como propedéutico, entre otros. El sistema fue creado a finales del año 2008 y entró en funcionamiento para el periodo 2009-2, en principio como un procedimiento de inscripción y reinscripción para los alumnos de ambos niveles. Hoy en día, el sistema, denominado "Ventanilla Virtual", ha integrado diversos servicios que están disponibles para los usuarios en cualquier momento y lugar que lo soliciten, esto es lo denominado "24 por 7", veinticuatro horas al día los siete días de la semana, proporcionando una mejora en los servicios.

Desafortunadamente, la creación de la "Ventanilla Virtual" no siguió una metodología de desarrollo debido a que se pretendió cubrir de manera rápida la necesidad de tener una extensión del sistema que ofreciera diferentes tipos de servicios vía Web. Como consecuencia, el desarrollo del mismo presentó algunos inconvenientes como la repetición de código, almacenamiento de datos inconsistentes, falta de documentación, entre otros.

El trabajo que aquí presento y que realicé como parte de mi trabajo en la Facultad de Música de la Universidad Nacional Autónoma de México (UNAM), consiste en el desarrollo y mantenimiento de las aplicaciones que brindan soporte por parte de la Secretaría de Servicios y Atención Estudiantil. En este contexto, para lograr que este sistema se pudiera modernizar en cuanto a su infraestructura tecnológica, fue prioritario conocerlo de manera interna: comprender y analizar cada uno de los trámites que cubría tanto para el personal docente, como para los alumnos así como también para los aspirantes.

Una vez que asimilé cada trámite que cubría el sistema, lo siguiente fue adentrarme a la forma en que éste fue programado; esto es, bajo el desarrollo Web de contenido dinámico con el lenguaje de programación PHP. Gracias a la formación que tuve en la Facultad de Ciencias, revisar el material requirió el conocimiento de:

- El análisis, diseño e implementación de aplicaciones; en lo que se refiere a la eficiencia, corrección y seguridad de las mismas. Estos conocimientos incluyen, pero no se restringen a: Diseño, Implementación y Manejo de Bases de Datos; Sistemas Operativos; Diseño, Instalación y Administración de Redes de Computadoras; Análisis, Diseño e Implementación de Aplicaciones Distribuidas o Concurrentes; Sistemas de Software; Programación Web; Sistemas Inteligentes para resolver problemas.
- Los fundamentos teóricos que me permiten analizar, diseñar e implementar aplicaciones locales, distribuidas o concurrentes, ya sea a través de redes de computadoras o al interior de las organizaciones.
- Los fundamentos teóricos de las Ciencias de la Computación que me permiten profundizar en temas de mi propia elección.

- Los fundamentos matemáticos que me permiten colaborar en el modelado y automatización de aplicaciones en otras ramas científicas.
- Lo relativo a la coordinación y participación en equipos de trabajo en el desarrollo de aplicaciones de cómputo.

Una vez que aprendí a manejar este nuevo lenguaje (llamado PHP), me dispuse a realizar el análisis de cada parte del sistema, para corregirlo o mejorarlo en su comportamiento de acuerdo a los requerimientos o necesidades de los diversos trámites que ofrece la Secretaría, tomando en cuenta que los cambios que tuviera que realizar, fueran únicamente internos y que no modificaran la vista del usuario.

Bajo estas circunstancias, se me asignó la tarea de investigar las posibles soluciones para la detección de los errores, darle una mayor agilidad a las diversas solicitudes, el ahorro de espacio y una mayor eficiencia del sistema. Todo esto con la libertad de reutilizar el código existente o generar uno nuevo, siempre y cuando se respetara la versión que manejan para el lenguaje PHP.

Las diversas soluciones que fueron implementadas, se llevaron a cabo con éxito. En el presente trabajo describo algunas de las soluciones propuestas para conseguir un funcionamiento más adecuado. En el primer capítulo presento una breve introducción sobre la Facultad de Música y el sistema llamado "Ventanilla Virtual". En el segundo capítulo, explico el manejo de la información que tiene en cuanto a su personal docente. En el tercer capítulo, muestro el principal problema de tener código repetido y código que es inservible al igual que la repetición de un mismo sistema. En el cuarto capítulo propongo prácticas que son de gran utilidad para la buena programación, donde toco diversos puntos que se deben tomar en cuenta a la hora de realizar un buen desarrollo de software y que es conveniente considerar en cualquier lenguaje de programación (en este caso para el lenguaje PHP), con la finalidad de crear un código que haga eficiente el funcionamiento del programa y evitar los malos hábitos en la programación, ya sea por la falta de tiempo, desidia, falta de conocimiento, falta de información, entre otros. Finalmente, presento mis conclusiones.

CAPÍTULO I
LA FACULTAD DE MÚSICA Y LA VENTANILLA VIRTUAL

La Facultad de Música (FaM) es responsable de formar profesionales en música:

- En la investigación etnomusicológica y musicológica.
- En la educación musical en los ámbitos de la enseñanza, promoción y extensión de la música.
- En la música de concierto en los campos de la composición, la interpretación, la docencia y la difusión.

Por su carácter público, la FaM ofrece servicios educativos y culturales a toda persona que cumpla con los requisitos establecidos tanto por la UNAM como los propios de la institución. Sus alumnos aseguran su permanencia en la Institución mientras manifiesten aprovechamiento académico y compromiso con sus estudios.

Por ser una institución nacional, abre sus puertas a todos los estudiantes del país y del extranjero; con sus actividades e investigaciones intenta prioritariamente atender las necesidades musicales del país. Por formar parte de una universidad autónoma, goza de plena libertad para organizarse, enseñar, investigar y difundir la cultura en el ámbito musical.

Tomando en cuenta lo anterior y en función de los distintos tipos de servicios que ofrece la Facultad de Música y considerando los diversos avances en las Tecnologías de la Información y la Comunicación (TIC) existentes, la Secretaría de Servicios y Atención Estudiantil de la Facultad de Música desarrolló un sistema llamado "Ventanilla Virtual" para satisfacer necesidades específicas que puedan ser atendidas en cualquier momento del día, con la finalidad de lograr una mejor respuesta a las diversas solicitudes de alumnos y profesores y evitar la dependencia del personal administrativo.

Este sistema se dirige a tres sectores: (I) Profesores, (II) Alumnos, (III) Aspirantes. Tales sectores se dividen, a la vez, en diversos procesos. A continuación se hace una descripción de los sectores y sus respectivos procesos.

I. Profesores.

Este módulo comprende de tres secciones:

1. Sistema de evaluación académica (EVA aaaa-d).
2. Gestión Académica.
3. Tutoría individual.

Las cuales se describen en seguida.

Sistema de evaluación académica de un semestre específico (EVA aaaa-d).

Es el sistema en el que el profesor realiza la evaluación académica de los alumnos (del nivel propedéutico) inscritos en cada uno de sus grupos, muy similar al proceso del Sistema Integral de Administración Escolar (SIAE) de la Dirección General de Administración Escolar (DGAE) para la evaluación académica que tienen los alumnos de nivel licenciatura.

La Secretaría de Servicios y Atención Estudiantil es la encargada de resguardar esta información y darle seguimiento al aprovechamiento académico de los alumnos, para verificar que cumplen con uno de los prerrequisitos que les permite continuar sus estudios a nivel licenciatura. El aaaa-d es para indicar el periodo semestral que está permitido evaluar, ya que existían 10 sistemas en total (los periodos comprendidos desde 2011-1 hasta 2015-2) en el todos tenían el mismo comportamiento (la solución para centralizar este sistema lo explico en el Capítulo III).

Gestión académica.

Este sistema también incluye una parte del personal administrativo. De acuerdo al nivel de permisos o puesto laboral que desempeña, puede acceder a las siguientes secciones:

1. Alumnos,
2. Profesores,
3. Procesos y
4. Mis grupos.

Éstas serán descritas a continuación.

Alumnos. El personal administrativo y parte del personal docente pueden realizar una búsqueda por medio de algún número de cuenta, nombre del alumno o por el nombre de la carrera o área. Una vez que hayan localizado al estudiante, puede obtener la siguiente información:

- Historia académica. Muestra su trayectoria escolar en el avance de créditos, asignaturas obligatorias, optativas, promedio, entre otros.
- Asignaturas. Pueden dar de alta, baja o cambio de grupos tanto en sus registros del periodo ordinario como extraordinario.
- Reposición de su contraseña. Es una solicitud en que muestra tanto la cuenta del usuario junto a la contraseña que debe coincidir para poder ingresar a la sección de alumnos en caso de que los alumnos la hayan olvidado.
- Suspensiones de estudios. Pueden eliminar o registrar todos los movimientos que el alumno haya solicitado para darse de baja temporal.
- Revalidación de asignaturas (nivel propedéutico). Si el alumno que ha ingresado tiene conocimientos previos o lo demostró por medio de exámenes de colocación, se le hace la revalidación de las asignaturas que haya aprobado el consejo.
- Solicitud de acta (nivel propedéutico). Verifica si el alumno tiene registrada un acta (ya sea de tipo ordinaria o extraordinaria) para ser evaluada o se le asigna un acta que esté disponible.
- Asignación de profesor (nivel propedéutico). Debido a que las asignaturas de preparación en el instrumento son individuales y para evitar que los alumnos se inscriban sin solicitar el permiso del profesor, esta sección crea una relación alumno-profesor para garantizar la inscripción y evitar que algún otro alumno se inscriba sin la autorización correspondiente.
- Horarios de asignaturas. Lista los horarios de cada asignatura con el profesor que imparte, para tener una localización de estos en caso que el alumno no tenga su horario o se necesite localizar al profesor.
- Imprimir registros. Imprime los comprobantes que requiera el alumno para confirmar su inscripción, trayectoria académica, registros de asignatura, entre otros.
- Datos de contacto. Se tiene información de contacto del alumno para informarle a cerca de becas, servicios sociales, conciertos, etcétera.

- Constancia de estudios. Si el alumno está inscrito actualmente, se le puede otorgar una constancia donde indica las fechas que comprenden el periodo. Por ejemplo, días de asueto, vacaciones, semana de exámenes.

Profesores. El personal administrativo y parte del personal docente pueden realizar una búsqueda por medio del nombre del profesor o por el nombre de la asignatura. Una vez que el profesor haya sido localizado, puede acceder a la siguiente información. Sus grupos, Crear acta, Asignar grupo, Inscritos, Datos de contacto.

- Grupos. Se pueden consultar todos los grupos (ya sea por nombre de la asignatura o el nombre del profesor) que se han dado de alta en el sistema.
- Acta (nivel propedéutico). Permite el registro de un acta para ser calificada a los alumnos que serán asignados.
- Creación de grupos. Este proceso principalmente lo maneja el jefe del Departamento de Servicios de Operación Logística y Estudiantil, en la que se encarga de crear, modificar o eliminar los grupos que serán ofertados durante el periodo escolar tanto ordinario como extraordinario.
- Inscritos. Esta opción únicamente les aparece a los profesores con grupos dados de alta en el sistema y que por lo menos tengan un alumno registrado, para así ver la lista de todos sus estudiantes. Si el profesor es parte del programa de tutorías, de la lista de sus alumnos puede acceder a los datos de contacto para mantener comunicación con ellos.
- Datos de contacto. Los profesores actualizan la información de los datos de contacto, para que la Secretaría mantenga actualizada la información y pueda enviar avisos relacionadas con la Facultad.

Procesos. Esta sección está dirigida al personal administrativo y se encarga de lo siguiente:

- Proceso de titulación. Se muestran a los alumnos que hayan comenzado con el proceso de titulación y se da un seguimiento de cada paso que el alumno debe realizar, así como la entrega de algún documento.
- Grupos. Pueden consultar todos los grupos (ya sea por nombre de la asignatura o el nombre del profesor) que se han dado de alta en el sistema para ser modificados, eliminarlos o dar de alta un nuevo grupo; así como ver la lista de los alumnos que se encuentran inscritos.
- Lista de usuarios. Se muestra un listado en el que se indica si es un profesor activo o inactivo. Es manejada únicamente por el jefe del Departamento de Servicios de Operación Logística y Estudiantil).
- Trayectorias escolares. Este es un proceso nuevo en el que determinados profesores y parte del personal académico analizan el nivel de aprovechamiento tanto de nivel licenciatura como propedéutico de acuerdo al avance de créditos y bajo el tiempo reglamentario o tiempo curricular, de esta manera se puede dar un seguimiento grupal (por generación) o particular (alumno por alumno), para determinar si se deben modificar los planes de estudio o las dinámicas que los profesores realizan.

Mis grupos. Esta sección es para el personal docente de la FaM y permite consultar los grupos que el profesor imparte en el periodo, indicándole el número de alumnos registrados y el horario de la asignatura, inscritos, tutoría, actualizar datos de contacto:

- Grupos. Pueden consultar todos los grupos que se han dado de alta desde el sistema y pueden ver la lista de los alumnos que se han inscrito en cada curso. Si el profesor está registrado dentro del programa de tutoría, entonces puede consultar la trayectoria académica de los alumnos registrados para darle seguimiento.
- Histórico grupal. Pueden consultar los grupos que han sido dados de alta en periodos pasados, incluyendo la lista de alumnos que se inscribieron.
- Datos de contacto. En esta sección pueden mantener actualizada su información de contacto para tener un medio de comunicación relacionado con la FaM.

Tutoría individual.

Con el objetivo de mejorar el desempeño escolar de los alumnos e incrementar la eficiencia terminal de su nivel correspondiente (propedéutico o licenciatura), un profesor tiene la libertad de decidir formar parte de este programa y analizar el seguimiento de las diversas problemáticas que impidan la conclusión de sus estudios, ya sea grupal o por alumno. De esta manera, puede crear propuestas que ayuden a mejorar los planes de estudio o sugerir una nueva forma de enseñanza.

II. Alumnos.

Este sector está formado por siete subsistemas que son detallados a continuación.

Servicios escolares.

Este sistema generaliza todas las solicitudes que realizaban en la ventanilla de servicios escolares y que, para agilizar el proceso se creó este sistema para que el alumno pueda solicitar (dependiendo de ciertas fechas) servicios como:

- **Actualización de datos personales.** Se les pide que mantengan actualizada su información personal (al menos en su correo electrónico) para informarles de trámites realizados, concursos internos o externos, convocatorias de beca, cursos, talleres, entre otros.
- **Consulta de evaluaciones (nivel propedéutico).** El alumno puede consultar sus evaluaciones tanto ordinarias como extraordinarias de las asignaturas que haya inscrito, se muestran el periodo pasado y en el que actualmente tengan cursado; de tal manera que puedan saber en tiempo real si el profesor ya realizó la evaluación correspondiente en caso de que en su historia académica no se vea reflejada.
- **Historial académico (nivel propedéutico).** Se les muestra el avance académico y las asignaturas que deben de llevar de acuerdo a su plan y semestre que llevan cursado.
- **Reposición de credencial (nivel propedéutico).** En caso de perder su credencial, el alumno imprimir el formato con su información previamente llenada para confirmar en ventanilla y que se le entregue una nueva.
- **Inscripción.** Todos los alumnos pueden registrar, modificar o eliminar las asignaturas que piden cursar dependiendo del calendario escolar. Una vez que tengan realizado el proceso, pueden ver sus asignaturas con un apartado en el que se les indica si su inscripción procede de acuerdo a los requisitos que requiera la asignatura.
- **Registro de audiciones.** Esto es para las áreas que deben cursar las asignaturas de Conjuntos Instrumentales o Conjuntos Orquestales, deben registrar su cita de audición para después realizar su inscripción de la materia (en caso de haber sido aprobado en la audición).
- **Registro de actividades extracurriculares.** Es un sistema de registro para asignaturas que no son parte de su plan de estudios pero que les puede servir para reafirmar conocimientos o realizar alguna actividad diferente como los talleres o materias adicionales.

- **Solicitud de cambio de área o segunda área (nivel propedéutico).** En caso de que el alumno quiera cancelar su inscripción y solicitar otra área que sea más de su agrado, la FaM tiene un procedimiento interno para que puedan realizar esté movimiento, así como los alumnos que hayan terminado una área y quieran estudiar otra que tenga relación. También hay un proceso interno para realizar la revalidación de ciertas materias y de ciertos exámenes para demostrar el interés de la nueva área.

Registro de cambio de nivel (nivel propedéutico).

Esta opción es para los alumnos que desean continuar con sus estudios a nivel profesional, mediante el registro para su examen y que cumpla con uno de los requisitos que establece la FaM para ingresar a la licenciatura que le corresponde, esta sección se les informa de cada paso que deben hacer para que su proceso de ingreso a la licenciatura se confirme.

Entrevista virtual y asignación de tutor.

En este sistema los alumnos pueden solicitar un tutor que les ayude a realizar diversas actividades o disciplinas para evitar la deserción y concluir exitosamente su aprovechamiento académico.

Calendarios escolares.

Indica todas las actividades tanto nivel licenciatura como propedéutico de acuerdo al periodo cursado, indicando las fechas que correspondencia a exámenes, periodo vacacional, inter semestral, inscripciones, entre otras.

Encuesta de evaluación.

Es un sistema que se encarga de la evaluación del desempeño docente de acuerdo al punto de vista de los alumnos.

Encuesta UNAM.

Sistema en línea que evalúa los servicios que otorga la UNAM a partir de la información que llenan los alumnos de licenciatura.

Reconocimiento académico.

Es una página en la que informa los alumnos que fueron propuestos para recibir la medalla "Gabino Barrera" de acuerdo al aprovechamiento académico.

III. Aspirantes.

El sector de Aspirante está integrado por Ingreso a propedéutico e Ingreso a licenciatura, las cuales se describen en seguida.

Ingreso a propedéutico.

Se encarga de dar seguimiento a la convocatoria de ingreso al nivel propedéutico, en ella se incluye el sistema de registro, información y fechas de aplicación que requiere para los exámenes de conocimientos musicales, etcétera.

Ingreso a licenciatura.

Se encarga de dar seguimiento a la convocatoria de ingreso al nivel licenciatura, en ella se incluye el sistema de registro, información y fechas de aplicación que requiere para los exámenes de conocimientos musicales, etcétera.

El sistema descrito anteriormente logró solucionar parte de las necesidades, aunque por el crecimiento de su población y la alta demanda de ingreso, requirió que se fueran añadiendo procesos sin tener una metodología de desarrollo, se puede ver del listado anterior que ciertas opciones tienen un comportamiento que es similar. Estas modificaciones trajeron como consecuencia el no poder centralizar su funcionamiento desde un solo código, debido a las repeticiones de archivos o principalmente la falta de documentación. Esto último, ha provocado la demora de centralizar el funcionamiento de cada componente, ya que a pesar de tener diversos archivos con el mismo nombre en el que contienen las funciones que necesita determinado sistema, no garantiza que las funciones definidas con el mismo nombre tengan el mismo comportamiento.

Los problemas en la documentación (con mala calidad o inexistente) de sus programas impide conocer en qué versión se encuentran, qué modificaciones (o correcciones) ha tenido o si su comportamiento es el mismo. Esto me obligó a tener que analizar cada uno de los sistemas, recolectar las funciones que aparentemente tienen puntos en común y verificar si su comportamiento es el mismo, para así poder agrupar y centralizar sus funcionalidades. Fue un inicio para comenzar a reducir algunas líneas del código y que a su vez generara una mayor funcionalidad más estable. Un ejemplo de esto, lo muestro en el Capítulo III al centralizar los siete sistemas de evaluación que el profesor tiene disponible y que realizan la misma necesidad, evaluar a los alumnos.

De la misma forma, un problema adicional es el almacenamiento de la toda su información, ya que al no tener un buen control de concurrencia en sus bases de datos, vuelve ineficiente el desempeño dentro del sistema, ya que se permite que el acceso a información compartida en algunos casos se vuelva de forma paralela, interfiriendo el buen manejo este ellas provocando conflictos o errores entre registros. Un ejemplo de esto, es en el sistema de inscripción de los alumnos, ya que si hay un cupo determinado menor a la demanda de los alumnos, se debe evitar que más de un alumno pueda registrarse al mismo tiempo en el grupo.

CAPÍTULO II
CONSISTENCIA EN DATOS

Con base en el capítulo anterior, para el acceso a la "Ventanilla Virtual" se consideraron tres tipos de usuarios que corresponden a las siguientes clasificaciones:

I. **Profesores.** Dirigida a todo el personal docente de la Facultad de Música tanto de nivel propedéutico como nivel licenciatura.
II. **Alumnos.** Dirigido a los estudiantes activos de la Facultad de Música, tanto nivel propedéutico como nivel licenciatura.
III. **Aspirantes.** Dirigido a quienes deseen participar en los procesos de selección internos de la FaM.

A los **profesores** se les asignó como nombre de usuario su Registro Federal de Contribuyentes (RFC) y una contraseña que genera el propio sistema para su acceso, así como un determinado nivel de permisos al momento de realizar sus solicitudes. De esta manera, el profesor accede a la información que requiere cuando lo necesite. Algunas de las opciones que el sistema proporciona son:

a) **Grupos.** Consultar los grupos que el profesor oferta de la o las asignaturas que imparte, tanto de nivel propedéutico como licenciatura. Se generan a partir de la clave de asignatura, una clave de grupo, nivel y su RFC.
b) **Inscripciones.** Consultar la lista de todos los alumnos que han solicitado su inscripción a alguno de sus cursos, en ambos niveles. Se genera a partir de la clave de asignatura, una clave de grupo, número de cuenta del alumno y su RFC.
c) **Evaluaciones.** Poder realizar la evaluación de los alumnos que hayan estado inscritos a alguna de las asignaturas, únicamente los de nivel propedéutico. Se generan con los datos de la inscripción más un número de folio único.
d) **Datos de contacto.** Permite actualizar su información personal y tener un medio de comunicación con el personal administrativo para casos relacionados con la Facultad. El almacenamiento de los datos personales está relacionada con su RFC.

Para las solicitudes anteriores, el RFC es un dato muy importante ya que en torno a él se agrupa la información de cada profesor, también porque la Secretaría, se encargar de informar al Sistema Integral de Administración Escolar (SIAE) de la Dirección General de Administración Escolar (DGAE), de los grupos ofertados durante el periodo, en un formato de archivo tipo texto con la siguiente estructura:

- Clave de grupo.
- Clave de asignatura.
- Turno (Sin turno, matutino, diurno, vespertino o nocturno).
- Clave del primer profesor (siendo su RFC).
- Clave del segundo profesor (siendo su RFC).
- Cupo.
- Tipo movimiento (ya sea si es una alta, baja o un cambio de grupo).

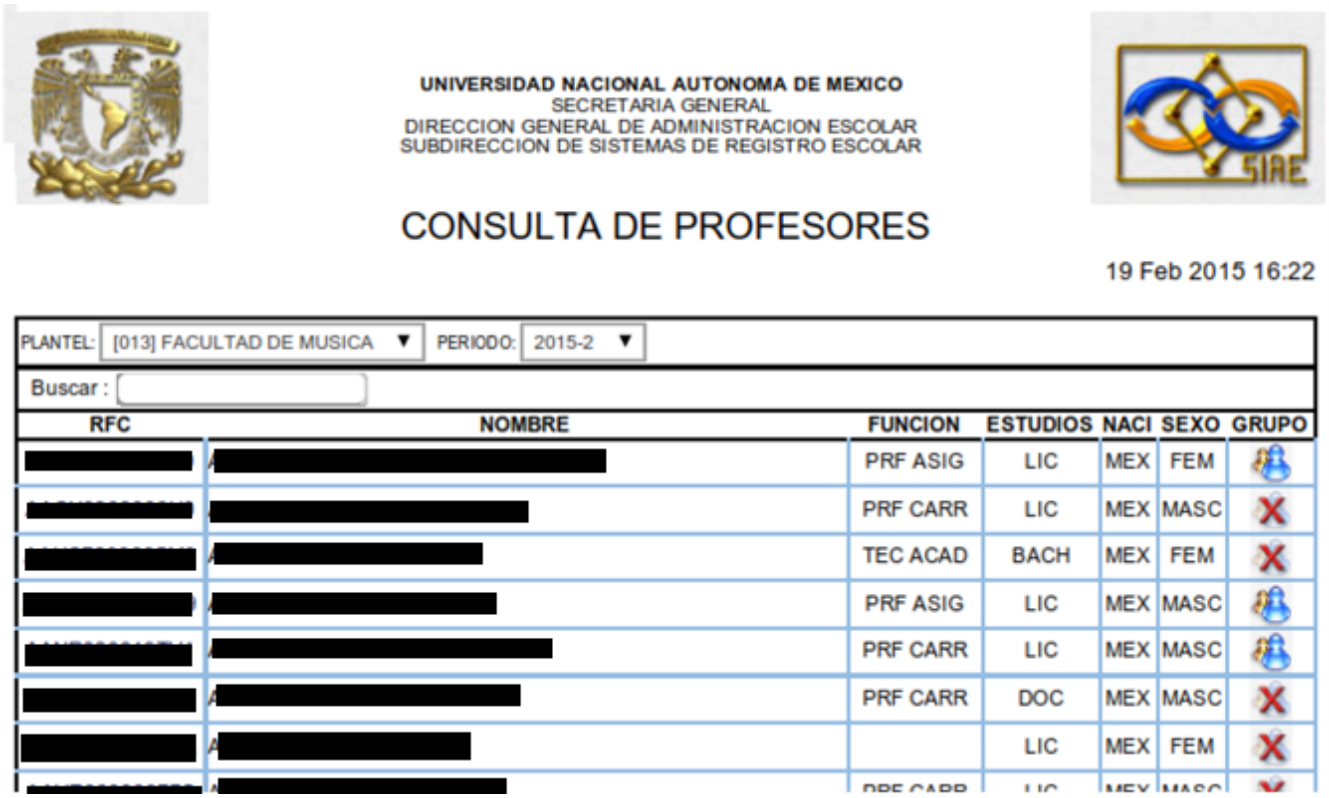
Este formato se crea únicamente para el caso de los grupos que se impartirán a nivel licenciatura; una vez creado, la Secretaría se encarga de enviarlo a la DGAE para realizar el registro y cotejo de los movimientos de los profesores durante el semestre en curso. En este punto surge un conflicto, ya que hay una diferencia en la información que tiene la Facultad de Música con la que tiene DGAE, principalmente al cotejar los RFC; este problema de cotejo se debe, entre otros, a errores al capturar este dato: falta de la Homoclave, confusión entre caracteres, que esté incompleto, o bien que el profesor no haya dado aviso de la actualización o modificación de su RFC.

Debido a esta diferencia en la información, era necesario cotejar manualmente los datos de los profesores y verificar que el RFC correspondiera al que la DGAE tenía dado de alta en su base de datos, y si no correspondía se sustituía. La realización de esta revisión era notoriamente lenta, debido a que el personal tenía que hacer una revisión de la lista de profesores que tienen en la Facultad con respecto a la lista que tenía en ese momento la DGAE, y que, a pesar de no ser efectivo y de correr el riesgo de que aún se presentaran algunos errores, a la Secretaría le correspondía repetir el proceso de revisión hasta que la DGAE pudiera confirmar que todos los registros eran correctos.

Para evitar la revisión y actualización constante durante cada semestre, diseñé una forma de darle solución automatizando dicho proceso, de forma tal que una sola persona tenga que cotejar si la comparación fue correcta o elegir la opción que le corresponde si la comparación tiene múltiples aproximaciones, reduciendo de esta manera la posibilidad de error, sobre todo por las diferentes intervenciones humanas. Elaboré una lista que funciona a manera de una transformación y que contiene el nombre del profesor, el RFC del mismo generado en la FaM y trasladarlo al valor del RFC del profesor que tiene dado de alta la DGAE, para así poder hacer simplemente el cambio de RFC que corresponde sin tener que modificar otro dato.

El planteamiento fue el siguiente: ¿de qué manera se pueden comparar dos listados compuestos por el RFC y el nombre de profesor de tal manera que sea lo más eficiente y con un error mínimo? El nombre del profesor es lo único que tengo de recurso para obtener una solución adecuada, por lo que el problema se reduce a realizar un proceso para relacionar la lista de RFC de profesores que tienen tanto la FaM como la DGAE. A la primera es fácil de acceder a la base de datos y pedir el RFC y nombre de cada profesor; para la segunda lo único que tuve a disposición fue un archivo en formato de documento portable, mejor conocido como PDF. La **Figura 1** se muestra la estructura y datos de este archivo.

Este archivo está compuesto por los siguientes datos: RFC, nombre completo del profesor, su función, nivel de estudios, lugar de nacimiento, sexo y una columna llamada grupo; de este documento lo único que necesito obtener era el RFC y el nombre completo del profesor. Lo siguiente fue realizar un programa que hiciera una lectura línea por línea del archivo para poder analizar, seleccionar y extraer de cada renglón únicamente los datos que fueran de utilidad. Sin embargo, la misma estructura del archivo PDF se mantenía al ser convertido a un archivo de texto plano con sólo copiar y pegar la información, cada renglón del PDF mantiene la información como una sola cadena separada por espacios en blanco; esto facilitó la forma de analizar la estructura del texto, así como evitar la creación de un lector de archivos para filtrar los datos innecesarios.


UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
SECRETARIA GENERAL
DIRECCION GENERAL DE ADMINISTRACION ESCOLAR
SUBDIRECCION DE SISTEMAS DE REGISTRO ESCOLAR

CONSULTA DE PROFESORES

19 Feb 2015 16:22

PLANTEL: [013] FACULTAD DE MUSICA PERIODO: 2015-2

Buscar :

RFC	NOMBRE	FUNCION	ESTUDIOS	NACI	SEXO	GRUPO
[redacted]	[redacted]	PRF ASIG	LIC	MEX	FEM	
[redacted]	[redacted]	PRF CARR	LIC	MEX	MASC	
[redacted]	[redacted]	TEC ACAD	BACH	MEX	FEM	
[redacted]	[redacted]	PRF ASIG	LIC	MEX	MASC	
[redacted]	[redacted]	PRF CARR	LIC	MEX	MASC	
[redacted]	[redacted]	PRF CARR	DOC	MEX	MASC	
[redacted]	[redacted]		LIC	MEX	FEM	

Figura 1. Listado de profesores de la FaM.

Así pues, del archivo en PDF tenía que filtrar cada línea de este texto y únicamente quedarme con el RFC y el nombre completo del profesor, omitiendo toda la información restante. Para llegar a este objetivo, me enfoqué en las opciones que aparecen en cada columna de la lista de profesores (RFC, NOMBRE, FUNCION, ESTUDIOS, NACI, SEXO y GRUPO). Por ejemplo, en la columna llamada FUNCION, se tienen las siguientes opciones:

1. **AYUD PRF.** Ayudante de profesor.
2. **PRF CARR.** Profesor de carrera.
3. **PRF ASIG.** Profesor de asignatura.
4. **TEC ACAD.** Técnico académico.
5. **AYUD O INV.** Ayudante o investigador.
6. Opción vacía en la que se desconoce su cargo.

De la misma forma para la siguiente columna que corresponde al grado de estudios del profesor (nombrada ESTUDIOS), se compone de las siguientes opciones:

1. **LIC.** Licenciatura.
2. **MTRIA.** Maestría.
3. **DOC.** Doctorado.
4. **BACH.** Bachillerato.
5. **DIPL.** Diplomado.
6. **TEC.** Nivel técnico.

Estas opciones (FUNCION o ESTUDIOS) forman un patrón en cada una de las líneas de cada profesor; esto quiere decir que cada línea a leer iniciará con la función a su cargo, o bien, con el grado de estudios (en caso de que la función a su cargo esté en blanco), de esta manera, cada renglón del archivo de texto plano da un patrón de inicio que debe ser eliminado o sustituido a partir de ese punto hasta el final de la línea. Esto fue posible hacerlo sin necesidad de crear un lector de archivos que realizara el filtro de toda la línea de caracteres.

Este filtro lo elaboré mediante el uso de expresiones regulares, las expresiones regulares, son un tipo de secuencias de caracteres que ayudan a formar, mediante un patrón de búsqueda para ser sustituible por alguna otra, la cual nos proveen una manera muy flexible de buscar o reconocer cadenas de texto. Al seguir esta mecánica mediante las opciones que tienen las columnas FUNCION y ESTUDIOS, el número de combinaciones posibles es finita y así pude definir lo siguiente:

Para la columna llamada FUNCION, únicamente aparecerán las opciones que listé anteriormente, entonces puedo decir que la secuencia se logra obtener de alguna de las siguientes opciones:

```
(AYUD PRF) | (PRF CARR) | (PRF ASIG) | (TEC ACAD) | (AYUD O INV)
```

De la misma forma para la columna ESTUDIOS, sólo aplica la aparición de una de las siguientes opciones:

```
(LIC) | (MTRIA) | (DOC) | (BACH) | (DIPL) | (TEC)
```

Para la columna FUNCION existe la posibilidad de que no esté definida, es decir, que aparezca la opción vacía y que el texto plano simplemente fuera tomado como un espacio en blanco; si esto llega pasar, deberá aparecer una de las opciones de la columna ESTUDIOS, ya que es la siguiente columna en cuestión y en ella no hay campo que quede vacío debido a que el personal docente debe tener al menos el bachillerato (BACH) como grado mínimo de estudios. Esto quiere decir que puede aparecer una opción de la columna FUNCION, o bien, que aparezca una opción de la columna ESTUDIOS; esto me indica que obligatoriamente puede aparecer la primera o segunda opción por omisión, el cual la expresión regular compuesta la puedo definir de la siguiente manera:

```
(FUNCION) | (ESTUDIOS)
```

Ahora, como entre el nombre y la columna FUNCION habrá un espacio en blanco al menos, (ya que anteriormente la columna FUNCION puede tener la opción vacía), entonces la secuencia de caracteres debe comenzar por lo menos con un espacio en blanco y es seguido de alguna de las columnas que ya mencioné anteriormente, de esta manera defino la siguiente expresión regular:

```
[ ]{1, }( (FUNCION) | (ESTUDIOS) )
```

Primero, con los paréntesis cuadrados indico el carácter especial que debe ir antes, el cual es un espacio en blanco; después indico la cantidad de espacios que esperamos, en este caso, lo que se encuentra en las llaves es una instrucción de la forma {n, m}, donde "n" y "m" son números naturales donde n <= m y lo cual nos indica que nuestra expresión puede tener como mínimo "n" repeticiones y a lo más hasta "m"; cuando la "m" no está definida o no se coloca ningún valor numérico (como lo que yo hice), quiere decir que no importa cuántas veces está repetida la expresión y que de esa

manera indicamos que al menos debe haber un espacio en blanco o más, seguido de alguna de las opciones de la columna FUNCION o la columna ESTUDIOS.

Por último, para poder descartar todas las opciones que tienen las demás columnas a partir de la expresión que construí y la cual me ayudó a identificar el inicio de la línea que se va a descartar, y que, para indicar las opciones de las columnas restantes que siguen después de esa expresión, son una serie de caracteres en letras mayúsculas y que también debo incluir el espacio en blanco, ya que pueden ser varias palabras, y como al menos debe existir un carácter o bien, una palabra, recurro al uso de los paréntesis cuadrados para indicar esa clase de caracteres que debo recibir (las letras del alfabeto y el espacio) y las llaves (para indicar el número de apariciones posibles), que anteriormente mencioné para poder abarcar cualquier secuencia sin saber qué opciones tiene cada columna de las que no hemos tomado en cuenta, obteniendo la siguiente expresión regular:

```
[A-ZÑ ]{1, }
```

De esta forma evito analizar las demás opciones y no tener que escribir cada una de las letras del alfabeto, uso el guión para indicar el rango de caracteres posibles; es decir, los que van de la A hasta la Z y en el que indico que deben ser mayúsculas, de esta manera se evita el tener que escribir cada una de las letras del alfabeto, le añado la letra "Ñ" y el espacio en blanco para la búsqueda, así abarco todas las combinaciones de palabras posibles y que estén separadas por espacios. Esta última definición se incluye a la expresión que ya construí para finalizar con la expresión regular:

```
[ ]{1, }( (FUNCION) | (ESTUDIOS) )[A-ZÑ ]{1, }
```

Una vez definida la expresión regular, se generaliza la cadena buscada para cada línea del archivo, por ejemplo, si tuviera la siguiente línea:

```
RFCS990099HHH ARZATE VARGAS DANIEL PRF CARR LIC MEX MASC
```

La expresión regular abarca todo lo subrayado cuando inicia la búsqueda, desde el espacio en blanco mediante el uso de "[ ]{1, }" y por la aparición de la opción "PRF CARR" (por la siguiente parte de la expresión en la que debe aparecer FUNCION o ESTUDIOS) y, por último, toda la línea de caracteres siguiente sin importar lo que este escrito debido a que coincide bajo la regla "[A-Z]{1, }". En cambio, para la siguiente línea del archivo:

```
RFCS009900HHH ALVARADO GOMEZ VICTOR LIC MEX FEM
```

Caemos en el segundo caso, en el que la columna FUNCION está vacía y por ende aparece la opción que corresponde a la columna ESTUDIOS, entonces, nuevamente en la expresión inicia la búsqueda desde lo que está subrayado en adelante, por el espacio en blanco ([ ]{1, }) y porque aparece la opción "LIC" que corresponde al de la columna ESTUDIOS (siendo correspondiente a la segunda parte de la expresión), y para finalizar, toda la línea de caracteres siguiente sin importar lo que esté escrito por medio de la tercera y última parte de la expresión: [A-Z]{1, }.

Así, al poder cubrir ambos casos (siendo todos los posibles), pude evitar el realizar un lector de archivos y fabricar este filtro; de esta manera realizo la búsqueda de todas estas líneas que tengan la estructura de mi expresión regular, para proceder a eliminar esa parte de la cadena que me es

innecesaria y que lo que resta es la cadena que tengo que procesar, la cual se compone de los datos que corresponden al RFC y el nombre del profesor en cuestión.

Con lo explicado anteriormente, al realizar una sustitución de la expresión regular por la cadena de una comilla simple y una coma (',), la siguiente parte del texto plano:

```
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA PRF ASIG LIC MEX FEM
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA PRF CARR LIC MEX MASC
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA TEC ACAD BACH MEX FEM
RFCSXXXXXXHHH VALLE ALVARADO CARMEN LIC MEX FEM
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA PRF CARR LIC MEX MASC
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA PRF CARR DOC MEX MASC
RFCSXXXXXXHHH VALLE ALVARADO CARMEN LIC MEX FEM
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA PRF CARR LIC MEX MASC
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA PRF CARR LIC MEX FEM
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA PRF CARR LIC EXT FEM
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA PRF CARR LIC MEX FEM
...
```

Se transforma a lo siguiente:

```
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA',
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA',
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA',
RFCSXXXXXXHHH VALLE ALVARADO CARMEN',
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA',
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA',
RFCSXXXXXXHHH VALLE ALVARADO CARMEN',
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA',
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA',
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA',
RFCSXXXXXXHHH GOMIZ ROMERO CLAUDIA',
...
```

De esta manera, por medio de la escritura de múltiples líneas de forma simultánea en NotePad++, se muestra lo siguiente:

```
'RFCSXXXXXXHHH' => 'GOMIZ ROMERO CLAUDIA',
'RFCSXXXXXXHHH' => 'GOMIZ ROMERO CLAUDIA',
'RFCSXXXXXXHHH' => 'GOMIZ ROMERO CLAUDIA',
'RFCSXXXXXXHHH' => 'GOMIZ ROMERO CLAUDIA',
'RFCSXXXXXXHHH' => 'GOMIZ ROMERO CLAUDIA',
'RFCSXXXXXXHHH' => 'GOMIZ ROMERO CLAUDIA',
'RFCSXXXXXXHHH' => 'VALLE ALVARADO CARMEN',
'RFCSXXXXXXHHH' => 'GOMIZ ROMERO CLAUDIA',
'RFCSXXXXXXHHH' => 'GOMIZ ROMERO CLAUDIA',
'RFCSXXXXXXHHH' => 'GOMIZ ROMERO CLAUDIA',
'RFCSXXXXXXHHH' => 'GOMIZ ROMERO CLAUDIA',
...
```

Al realizar estas modificaciones adicionales (tanto la comilla simple del inicio " ' " como la cadena " ' => ' "), se crea la sintaxis que se maneja en PHP para los arreglos asociativos (ya que tanto para los arreglos indexados como asociativos no hace distinción alguna). Un arreglo en PHP se define como un mapa ordenado; un mapa es un tipo de datos que asocia valores con claves. Este tipo es optimizado para varios usos diferentes; puede ser usado como una matriz real, una lista (también conocida como vector), una tabla asociativa (la implementación de un mapa), un diccionario, una colección, una pila, una cola y posiblemente más tipos, ya que los valores de un arreglo también pueden ser otros arreglos, árboles y también son posibles arreglos multidimensionales. Así que, con la ayuda del constructor *array()*, formo parejas de la forma clave => valor, donde la clave corresponde al RFC y el valor es el nombre del profesor.

```
// Defino mi variable de tipo array()
$profesoresDgae = array(
    'RFCSXXXXXXHHH' => 'GOMIZ ROMERO CLAUDIA',
    'RFCSXXXXXXHHH' => 'VALLE ALVARADO CARMEN',
    'RFCSXXXXXXHHH' => 'GOMIZ ROMERO CLAUDIA',
     ...
    'RFCSXXXXXXHHH' => 'GOMIZ ROMERO CLAUDIA',
    'RFCSXXXXXXHHH' => 'VALLE ALVARADO CARMEN'
);
```

Debido a que el documento en PDF de la DGAE contiene 357 profesores registrados adscritos a la Facultad de Música, decidí convertir directamente la información en un arreglo, ya que al tener que almacenar los datos en una tabla, al final tengo que realizar la consulta y almacenarla en este tipo de dato para después procesar cada registro.

De esta manera obtuve los registros del personal docente de la DGAE y a la vez realicé el mismo procedimiento para los registros que tiene la Facultad de Música, esto me resultó mucho más sencillo gracias al Sistema de Administración de Bases de Datos (DBMS por sus siglas en inglés) llamado MySQL. Este motor de base de datos elegido por la Secretaría de Servicios y Atención Estudiantil me facilitó realizar una consulta de todos los registros de los profesores que hayan sido dados de alta y sólo pedir tanto su RFC como su nombre completo, para después pasar a almacenarlo en otro arreglo.

Debo mencionar que con el **Código 1** que expongo en el **Anexo**, a pesar de tener algunos comentarios en el bloque de código, es fundamental producir la estructura de documentación correcta para evitar analizar línea por línea el bloque de instrucciones para conocer su comportamiento, incluyendo también si tiene algún fallo o error de ejecución para casos específicos. Un código debidamente documentado ayuda que la persona que necesite utilizar una determinada biblioteca de clases o funciones tendrá toda la información necesaria: qué hace cada elemento y cómo se utiliza. No necesita acceder al código fuente.

Existen algunas herramientas que permiten generar documentación de forma automática a partir del código fuente. Javadoc es la herramienta estándar en Java, de tal manera que para PHP una de las herramientas más utilizadas es phpDocumentor (www.phpdoc.org).

Ahora, la función *mysql_fetch_array()* de PHP devuelve cada una de las filas del resultado de la consulta anterior hasta que devuelva un valor nulo. Al mismo tiempo, para aprovechar estas llamadas mediante un ciclo *while*, también recurrí a las funciones *array_search()* y *array_keys()*, la primera función busca un valor determinado en un arreglo y devuelve la clave que le corresponde en caso de que dicho valor exista y la segunda me indica si existen al menos dos repeticiones del valor buscado, evitando el problema de la existencia de los homónimos; es decir, el siguiente arreglo está definido con los siguientes valores:

```
$arreglo = Array(
        'RFCSXXXXXXHH1' => 'GOMIZ ROMERO CLAUDIA',
        'RFCSXXXXXXHH2' => 'VALLE ALVARADO CARMEN',
        'RFCSXXXXXXHH3' => 'AVALOS ARAGON HUMBERTO',
        'RFCSXXXXXXHH4' => 'ORTIZ PRADO JOSE'
);
```

Al querer buscar al profesor con el nombre AVALOS ARAGON HUMBERTO, al ejecutar la instrucción *array_search('AVALOS ARAGON HUMBERTO', $arreglo)* me devuelve la clave 'RFCSXXXXXXHH3' dado que el arreglo *$arreglo* contiene el valor 'AVALOS ARAGON HUMBERTO'; en caso no existir el valor a buscar, la función devuelve la constante booleana *FALSE*. Para el uso de *array_keys($arreglo, 'AVALOS ARAGON HUMBERTO')*, regresa un arreglo con todas las claves que le corresponden a ese valor (en este caso sólo devuelve un arreglo de un solo elemento).

De esta manera hice un comparativo en que cubro la posibilidad de que haya nombres escritos exactamente iguales siempre y cuando no haya homónimos al querer buscarlo, con esto me ahorro tiempo de ejecución al realizar este procedimiento además del uso de memoria al evitar el almacenar nombres adicionales. De esta forma, con el **Código 2** que expongo en el **Anexo** obtengo el segundo listado que corresponde a los profesores de la Facultad de Música y que necesité para la tercera y última parte.

Una vez concentrada esta información (el listado de RFC de la Facultad de Música y los RFC de la DGAE), tuve que crear una nueva lista en la que obtendría la relación de los dos RFC con respecto al nombre del profesor que obtuve al hacer la comparación de las dos listas anteriores; de esta manera hay un ahorro de tiempo y errores al automatizar este trabajo, ya que el procedimiento que realizaba la Secretaría, era quien realizaba la búsqueda y sustitución del RFC de su base de datos por el RFC de la DGAE, era el personal administrativo, generando un gasto de tiempo innecesario y con una alta probabilidad de generar errores al realizar la sustitución.

Este procedimiento es el punto principal del problema, debido a que al realizar el comparativo de la cadena completa que tienen los nombres no es efectiva, ya que también la captura de los nombres de ambas listas contiene la misma problemática (como los RFC). Problemas como:

- Faltas de ortografía.
- Caracteres, pronombres o artículos adicionales o faltantes.
- Tildes o acentos adicionales o faltantes.
- Abreviaturas.
- Nombres o apellidos adicionales o faltantes.
- Homónimos.

Para solucionar estas cuestiones, utilizo una herramienta basada en cadenas; en específico, se aplicó la solución del problema de encontrar la subsucesión común más larga, LCS[1]. La cual se describe en la siguiente sección.

Subsucesión común más larga, LCS.

En esta sección explico el problema de encontrar la subsucesión común más larga, donde el objetivo es comparar dos cadenas y tiene como finalidad el determinar su similitud mediante las siguientes dos formas:

- Si una cadena es subcadena de la otra.
- Encontrar una tercera cadena en la cual se parezcan a las dos cadenas anteriores.

El problema se concentra en encontrar la subsecuencia común más larga a todas las secuencias de un conjunto de secuencias, de las cuales por lo general llegan a ser sólo dos.

Cabe hacer hincapié que esto es muy diferente al problema de la subcadena común más larga, ya que a diferencia de las subcadenas, las subsecuencias no están obligadas en tener una cierta posición fija o que sea consecutiva dentro de las secuencias a comparar (esto es muy importante de remarcar, porque si alguna de las cadenas le falta o tiene un carácter de más, no se verá afectada la solución); de esta manera logré controlar aquellos errores de captura que no hayan sido verificados. La solución fue tomar cada uno de los registros de una lista y obtener todas las coincidencias con respecto a la otra lista en relación a su subsucesión máxima mediante un cierto margen mínimo de error, para así descartar las coincidencias que no tengan ni una duda en que se parezcan y así enfocarme en encontrar el posible nombre más cercano de la lista. Generalizando el procedimiento y para comenzar a explicarlo requiero lo siguiente:

Definición. Una subsucesión de una secuencia dada es sólo la secuencia dada con cero o más elemento omitidos. Formalmente, dada una sucesión $X = \langle x_1, x_2, \ldots, x_m \rangle$, otra sucesión $Z = \langle z_1, z_2, \ldots, z_k \rangle$ es una subsucesión de X si existe una secuencia estrictamente creciente $\langle i_1, i_2, \ldots, i_k \rangle$ (no necesariamente contiguos) de índices de X tales que para toda $j = 1, 2, \ldots, k$, se tiene que $x_{i_j} = z_j$. Por ejemplo, $Z = \langle B, C, D, B \rangle$ es una subsucesión de $X = \langle A, B, C, B, D, A, B \rangle$ con la secuencia de índices correspondiente $\langle 2, 3, 5, 7 \rangle$.

Ahora, dadas X y Y sucesiones, se dice que Z es una subsucesión común de X y Y, si Z es subsucesión de X y de Y. Por ejemplo, si $X = \langle A, B, C, B, D, A, B \rangle$ y $Y = \langle B, D, C, A, B, A \rangle$, entonces la sucesión:

$\langle B, C, A \rangle$ es una subsucesión común de X y de Y (aunque no es la más larga),
$\langle B, C, B, A \rangle$ también lo es,
$\langle B, C, A, B \rangle$ y $\langle B, D, A, B \rangle$ también lo son y no hay una más larga.

Para la generalización y mediante el uso de la programación dinámica, tengo lo siguiente:

Entrada de datos: $X = \langle x_1, x_2, \ldots, x_m \rangle$ e $Y = \langle y_1, y_2, \ldots, y_n \rangle$
Salida de datos: La longitud máxima de una subsucesión más larga de X y de Y.

[1] LCS: Siglas en inglés de *Longest Common Subsequence*.

Para la primera etapa se caracteriza la estructura óptima mediante el teorema de subestructura óptima de una LCS. Para el teorema se requiere tomar en cuenta que para una sucesión $X = \langle x_1, x_2, \ldots, x_m \rangle$, se define el prefijo iésimo de X, para $i = 0, 1, \ldots, m$, como $X_i = \langle x_1, x_2, \ldots, x_i \rangle$. Por ejemplo, si $X = \langle A, B, C, B, D, A, B \rangle$, entonces $X_4 = \langle A, B, C, B \rangle$ y X_0 es la sucesión vacía.

Teorema de subestructura óptima de una LCS.

Sea $X = \langle x_1, x_2, \ldots, x_m \rangle$ y $Y = \langle y_1, y_2, \ldots, y_n \rangle$ dos sucesiones y sea $Z = \langle z_1, z_2, \ldots, z_k \rangle$ alguna LCS de X y de Y.

1. Si $x_m = y_n$, entonces $z_k = x_m = y_n$ y Z_{k-1} es una LCS de X_{m-1} y Y_{n-1}
2. Si $x_m \neq y_n$ y $z_k \neq x_m$ entonces Z es una LCS de X_{m-1} y Y
3. Si $x_m \neq y_n$ y $z_k \neq y_n$ entonces Z es una LCS de X e Y_{n-1}

Con esto, la estructura de una solución óptima al problema LCS, incluye soluciones óptimas a subproblemas del mismo problema.

Para la segunda etapa, mediante una solución recursiva se examina cada subproblema para encontrar la longitud de la LCS, es decir, la longitud de una sucesión $X = \langle x_1, x_2, \ldots, x_k \rangle$ de k elementos se representa como $long(X) = k$. Por medio de la representación del iésimo prefijo se tiene que:

1. Si $x_m = y_n$, entonces $long\big(\mathrm{LCS}(X, Y)\big) = long\big(\mathrm{LCS}(X_{m-1}, Y_{n-1})\big) + 1$,
2. Si $x_m \neq y_n$, entonces $long\big(\mathrm{LCS}(X, Y)\big) = \max\{long\big(\mathrm{LCS}(X_{m-1}, Y)\big),\ long\big(\mathrm{LCS}(X, Y_{n-1})\big)\}$

Ahora, sea C una matriz tal que si defino a la celda $C_{[i,j]} = long\left(\mathrm{LCS}(X_i, Y_j)\right)$, bajo la siguiente fórmula recursiva:

$$C_{[i,j]} = \begin{cases} 0, & \text{si } i = 0 \text{ ó } j = 0 \\ C_{[i-1,j-1]} + 1, & \text{si } i, j > 0 \text{ y } x_i = y_j \\ \max\{C_{[i-1,j]}, C_{[i,j-1]}\}, & \text{si } i, j > 0 \text{ y } x_i \neq y_j \end{cases}$$

Esta fórmula recursiva expresa que dependiendo de la condición, es el subproblema que debe considerarse para ir calculando la longitud de las posibles LCS de las sucesiones. Aplicando este proceso mediante dos sucesiones $\mathrm{X} = \langle \mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_\mathrm{m} \rangle$ y $\mathrm{Y} = \langle \mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_\mathrm{n} \rangle$ como entradas, realizo lo siguiente:

1. Calculo la longitud de las sucesiones X y Y a las que llamo m y n respectivamente.

2. Inicializo un arreglo llamado M de tamaño $m \times n$, tal que $M_{[i,j]} = 0$ para toda $\mathrm{i} = 0, 1, \ldots, \mathrm{m}$ y $\mathrm{j} = 0, 1, \ldots, \mathrm{n}$.

3. Se calcula la longitud de la subsucesiones posibles al recorrer el arreglo M y usando a X y Y bajo la siguiente condición:

    a. Si el elemento $x_i = y_j$, entonces asignamos a $\mathrm{M}_{[i,j]} = 1 + \mathrm{M}_{[i-1,j-1]}$.

    b. Si no, entonces asignamos a $\mathrm{M}_{[i,j]} = max\{\mathrm{M}_{[i-1,j]}, \mathrm{M}_{[i,j-1]}, \mathrm{M}_{[i-1,j-1]}\}$.

    Una vez que se asignen los valores correspondientes a los índices i y j de M, devuelvo el arreglo.

4. Construyo una LCS (ya que puede no ser la única) a partir del valor de la casilla inferior derecha, es decir, el elemento del arreglo $\mathrm{M}_{[\mathrm{m}-1,\mathrm{n}-1]}$. Mientras el valor de $\mathrm{M}_{[i,j]} > 0$:

    a. Guardo el valor máximo entre $\mathrm{M}_{[\mathrm{i}-1,\mathrm{j}]}$, $\mathrm{M}_{[\mathrm{i}-1,\mathrm{j}-1]}$ o $\mathrm{M}_{[\mathrm{i},\mathrm{j}-1]}$.

    b. Si el valor máximo es $\mathrm{M}_{[\mathrm{i}-1,\mathrm{j}-1]}$ entonces:

        i. Si $\mathrm{M}_{[\mathrm{i}-1,\mathrm{j}-1]} = \mathrm{M}_{[\mathrm{i},\mathrm{j}]} - 1$, entonces nombro a z_k al elemento x_i o y_j (siendo el mismo), con $k \geq 0$ y $k \leq m$ (al elegir x_i) o $k \leq n$ (al elegir y_j).

        ii. Decremento tanto el valor de i y j en uno.

    c. Si el valor máximo es $\mathrm{M}_{[\mathrm{i}-1,\mathrm{j}]}$ entonces decremento tanto el valor de i en uno.

    d. Si el valor máximo es $\mathrm{M}_{[\mathrm{i},\mathrm{j}-1]}$ entonces decremento tanto el valor de j en uno.

    Cuando el ciclo termine, devuelvo la construcción de $\mathrm{Z} = \langle \mathrm{z}_1, \mathrm{z}_2, \ldots, \mathrm{z}_\mathrm{k} \rangle$ como una LCS.

De esta manera determino una LCS correspondiente a las sucesiones X y Y elegidas. Para ilustrar su funcionamiento, se puede observar en el siguiente ejemplo:

Sea $X = 10010101$ y $Y = 010110110$, cadenas de ceros y unos de las cuales se va a determinar la subsecuencia común más larga. En la que de acuerdo a lo anterior, se crea una matriz de tamaño $\text{longitud}(X) = 8 \times 9\ \text{longitud}(Y)$. Y se construye la siguiente matriz:

	Y_j	0	1	0	1	1	0	1	1	0
X_i	0	←0	←0	←0	←0	←0	←0	←0	←0	←0
1	↑0									
0	↑0									
0	↑0									
1	↑0									
0	↑0									
1	↑0									
0	↑0									
1	↑0									

De la cual, al realizar los tres casos posibles y comenzando desde el lado inferior derecho de nuestra matriz, de la cual tiene un valor de seis (la longitud de la subcadena).

	Y_j	0	1	0	1	1	0	1	1	0
X_i	0	←0	←0	←0	←0	←0	←0	←0	←0	←0
1	↑0	↑←0	↖1	←1	←1	←1	←1	←1	←1	←1
0	↑0	**↖1**	**←1**	↖2	←2	←2	←2	←2	←2	←2
0	↑0	**↖1**	←1	**↖2**	←2	←2	↖3	←3	←3	←3
1	↑0	↑0	**↖2**	←2	**↖3**	**←3**	←3	↖4	←4	←4
0	↑0	↖1	←1	**↖3**	←3	←3	**↖4**	←4	←4	↖5
1	↑0	↑0	↖2	←2	**↖4**	**←4**	←4	**↖5**	**←5**	←5
0	↑0	↖1	←1	↖3	←3	←3	**↖5**	←5	←5	**↖6**
1	↑0	↑0	↖2	←2	↖4	←4	←4	**↖6**	←6	←6

Una vez construida la tabla, el camino sombreado indica las dos subcadenas que son las soluciones con un tamaño de 6 caracteres, de tal manera que si realizamos la comparación de cada una de las soluciones:

Solución A:

```
X  = 1 0 0 1 0 1   0 1
Z  =     0 1 0 1   0 1
Y  =     0 1 0 1 1 0 1 1 0
```

Solución B:

```
X  = 1 0   0 1   0 1   0 1
Z' =   0   0 1   0 1   0
Y  =   0 1 0 1 1 0 1 1 0
```

Se puede observar que tanto la solución A y B corresponden a ser sub-sucesiones de X y de Y. Por lo tanto, Z y Z' son sub-sucesiones máximas de X y de Y.

Una vez concretada esta última parte del algoritmo, al obtener la subsecuencia de longitud máxima de la comparación entre las dos cadenas, al realizar este proceso para el caso de la comparación de todos los nombres del personal académico de la Facultad de Música con respecto a los que tiene la DGAE, se realiza lo siguiente:

Calculo la longitud de las tres cadenas.

```
tamaño1 = longitud(cadena1)
tamaño2 = longitud(cadena2)
tamaño3 = longitud(subcadena)
```

Elijo la cadena más larga como pivote y calculo su porcentaje de aproximación con respecto a la longitud de la subcadena obtenida.

```
if(tamaño1 <= tamaño2)
    porcentaje = (tamaño3 * 100) / tamaño1
else
    porcentaje = (tamaño3 * 100) / tamaño2
```

Mediante una regla de tres calculo el porcentaje de coincidencia más cercano de la subsecuencia en relación a la primera o segunda cadena, se toma como un mínimo de 85% (aunque este puede variar dependiendo del resultado esperado) de cercanía para poder descartar todas las demás cadenas e ir seleccionando la cadena con más alto porcentaje. Una vez concluido este procedimiento, es cuando se necesitaría de una persona quien pueda verificar qué tan efectivo es este análisis y dar sus debidas observaciones necesarias para la corrección o mejora del proceso. Aceptado el análisis que se realice de ambos listados, entonces se procede a almacenar estas comparaciones en una nueva tabla la cual contiene finalmente la transformación adecuada.

El proceso realizado para estos dos listados a modo de ejemplo de lo que ocurrió entre todas estas comparaciones, fue como en el caso del nombre del director de la Facultad de Música cuando en la cadena sobran o faltan caracteres, en el cual viene registrado como VIESCA Y TREVIÑO FRANCISCO JOSE para la FaM, y para la DGAE está como VIESCA TREVIÑO FRANCISCO JOSE. De donde al construir la subsucesión de la cadena más larga tenemos lo siguiente:

Nombre registrado en la FaM: VIESCA Y TREVIÑO FRANCISCO JOSE

Subsucesión obtenida: VIESCA TREVIÑO FRANCISCO JOSE

Nombre registrado en la DGAE: VIESCA TREVIÑO FRANCISCO JOSE

La subsucesión tiene un 100% de coincidencia con respecto al nombre que se tiene registrado tanto en la DGAE como en la FaM y, por ende, ya no hay necesidad de realizar más comparativos con los demás nombres, ya que el algoritmo ha encontrado la mejor opción. Otro caso es el de la profesora cuyo nombre es: MONICA DEL PILAR DEL AGUILA CORTES, de donde la Facultad de Música la tiene registrada como DEL AGUILA CORTES MONICA DEL PILAR y para la DGAE está registrada como AGUILA CORTES MONICA DEL PILAR DEL, gracias a la solución propuesta podemos observar que la subsucesión que coincide entre estos dos nombres es la siguiente:

Nombre en la FaM: DEL AGUILA CORTES MONICA DEL PILAR

Subsucesión obtenida: AGUILA CORTES MONICA DEL PILAR

Nombre en la DGAE: AGUILA CORTES MONICA DEL PILAR DEL

Donde la subsucesión obtenida, tiene un margen de 88.23% de coincidencia en ambos nombres (tanto de la FaM como la DGAE). Finalizado el proceso, se creó una tabla en la cual se usa a modo de transformación para lograr sustituir los RFC de la FaM por los de la DGAE. Esta modificación ayuda al momento de generar los reportes de los grupos ofertados para el periodo actual, y que en caso de que alguna de las dependencias tenga algún registro nuevo, se puede realizar el mismo proceso para verificar sus datos y ahorrar tiempo. Teniendo un resultado como se muestra en la **Figura 2**.

[redacted]D2	BAÑUELAS AMPARAN ROBERTO	[redacted]3	BAÑUELAS AMPARAN ROB
[redacted]L17	CORONA ALCALDE ANTONIO BENIGNO FELIPE	[redacted]HS8	CORONA ALCALDE ANTON
[redacted]I8	GONZALEZ CRUZ MARIA TERESA	[redacted]18	GONZALEZ CRUZ MARIA TE
[redacted]J6A	GUTIERREZ LOPEZ MIJAEL	[redacted]J64	GUTIERREZ LOPEZ MIJAEL
[redacted]5M8	MURUA MARTINEZ SALDAÑA VERONICA	[redacted]B60	MURUA MARTINEZ SALDAÑ
[redacted]N5	PEREZ CASTRO ANA BELLA	[redacted]N4	PEREZ CASTRO ANA BELL
[redacted]552	PIMENTEL GUERRERO GUSTAVO ADOLFO	[redacted]7I3	PIMENTEL GUERRERO GUS
[redacted]PK6	RAMOS ZEPEDA LUIS HUMBERTO	[redacted]DE2	RAMOS ZEPEDA LUIS HUM
[redacted]	REYES ZAMORANO ALEJANDRA	[redacted]E42	REYES ZAMORANO ALEJA

Figura 2. Tabla comparativa entre la lista de la FaM respecto a la lista de DGAE.

CAPÍTULO III
REDUNDANCIA DE CODIGO

La solicitud de ingreso a nivel licenciatura en la Facultad de Música, no sólo requiere del proceso de selección que la UNAM realiza. Adicionalmente, se pide como requisito de ingreso haber cursado un bachillerato musical (con duración de tres años o su equivalente), en el que el aspirante debe tener ciertos conocimientos y habilidades que se requieren de acuerdo al perfil de ingreso establecido en el plan y programa de estudio de la licenciatura a elegir.

En caso de que el aspirante no pueda comprobar o cubrir los conocimientos solicitados, la Facultad de Música ofrece un programa llamado Ciclo Propedéutico en Música, en modalidad escolarizado con una duración de tres años (equivalentes a seis periodos semestrales) en cualquiera de sus seis áreas: Canto, Composición, Etnomusicología, Educación Musical, Instrumentista y Piano; mismas que corresponden a las seis licenciaturas que la Facultad de Música oferta.

La Secretaría de Servicios y Atención Estudiantil es responsable de este programa y se encarga de manejar toda la información relacionada con el aspirante, el alumno inscrito desde su ingreso, estancia o hasta su egreso. Para llevar a cabo de manera ágil la forma de evaluar a los alumnos, a partir del periodo 2011-1 se creó un sistema para que el personal académico que imparte asignaturas a nivel propedéutico tuviera la oportunidad de realizar esta acción en cualquier lugar y momento del día desde su computadora.

Este sistema fue llamado *EVA 2011-1* y se orientó a la atención de Servicios Escolares a través de la VENTANILLA VIRTUAL, facilitando el que los alumnos pudieran consultar en cualquier momento del día si su profesor ya había evaluado y que calificación obtuvo en la asignatura inscrita. El funcionamiento es el siguiente:

1. El profesor inicia sesión mediante su RFC y su contraseña.

Figura 3. Inicio de sesión.

2. Selecciona el tipo de examen a calificar (ordinario o extraordinarios) y aparecerán todos los grupos que impartió durante el semestre.

Figura 4. Selección de la asignatura.

3. Una vez seleccionado el grupo, selecciona a cada uno de los alumnos para determinar su evaluación durante el curso.

Figura 5. Evaluación de alumnos.

4. Ya que el profesor termine de calificar a todos los alumnos, debe de cerrar el acta mediante su rasgo electrónico de identificación (algo equivalente a una contraseña, dada de alta por ellos).

Figura 6. Confirmación del cierre de acta.

5. Una vez que el profesor confirme el cierre de su acta, podrá imprimir el comprobante que debe dejar en la Ventanilla de Servicios Escolares Propedéutico para concluir con el proceso.

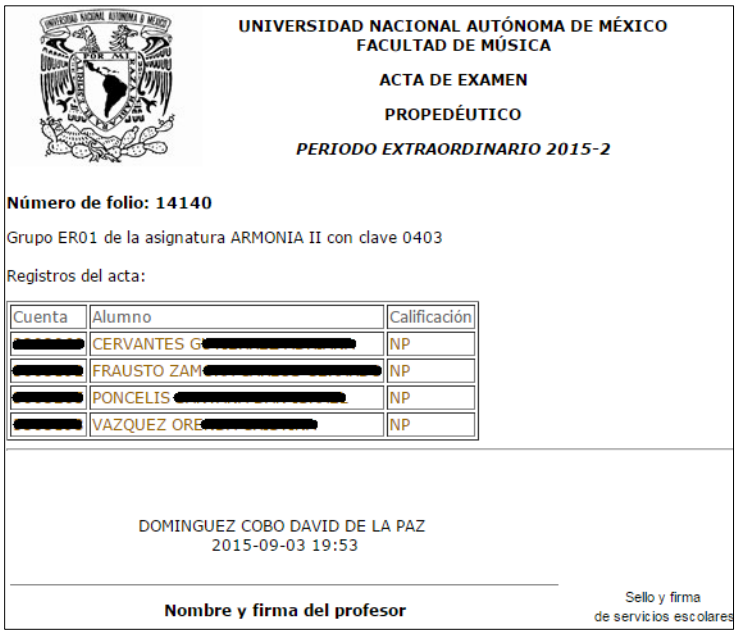


Figura 7. Impresión de la confirmación del cierre de acta.

Debido al éxito de esta nueva forma de evaluación, durante cada uno de los periodos siguientes se fueron generando los sistemas de evaluación: *EVA 2011-2*, *EVA 2012-1*, ..., *EVA 2015-1* hasta *EVA 2015-2*, a pesar de haber sido aceptado por el personal docente sin muchos problemas ya que la

mecánica de evaluar es similar al sistema que tiene la DGAE-SIAE a nivel licenciatura, agilizando este proceso. Como consecuencia de lo anterior, se va generando una repetición de código constante debido a que cada uno de los sistemas se copia exactamente igual salvo que lo único que cambia es el periodo.

Cada que se crea un sistema de evaluación (siendo un total de dos por año debido a que se crea uno por cada semestre), se reutilizan una serie de carpetas que contienen imágenes (de tipo JPG y GIF) y archivos que realizan el proceso de calificar (de tipo HTML, PHP, JavaScript y CSS), provocando que utilicen un espacio de memoria del servidor Web. Si la Facultad de Música estuviera pagando por un límite de espacio, el cobro se estaría incrementando conforme al paso del tiempo, siendo que varios de estos sistemas que ya cumplieron con su propósito (al igual que la DGAE-SIAE tienen un límite de tiempo para calificar y rectificar en caso de equivocarse).

Otro de los problemas para el caso de la programación, es que si se necesitara realizar alguna modificación como la vista de la página o la mejora en la programación para cualquiera de los periodos activos, esto conlleva a ir a cada carpeta de cada uno de los sistemas de evaluación para realizar los cambios; de la misma forma en cuanto a la programación en PHP, algunos archivos contienen funciones repetidas y errores de convención al programar como: la indentación, documentación del código, variables sin usar, reescribir funciones existentes, convenciones al nombrar variables y funciones.

Por ejemplo, en cada uno de los sistemas de evaluación existe un archivo que contiene una serie de funciones en la que los usuarios realizan consultas para determinadas acciones, de las cuales ninguna tiene su correcta documentación, hay un total de 23 funciones que son:

```
1. esUsuario ( $ncU, $pwU );
2. esProfesor ( $rfcU, $pwU );
3. OV1Usuario ( $ncU, $pwU );
4. OV2inscritas ( $ncU, $planU );
5. OV3inscritas ( $ncU, $planU );
6. OV2conrepertorio ( $ncU, $planU );
7. OV2asignaturas ( $nivelU, $planU );
8. OV3asignaturas ( $nivelU, $planU );
9. OV2cardex ( $ncU, $planU );
10. OV2grupos ( $nivelU, $claveU );
11. OV3grupos ( $nivelU, $claveU );
12. OV1registrogrupo ( $nivelU, $bloqueU, $grupoU );
13. OV1ordregistrogrupo ( $claveU, $bloqueU, $grupoU );
14. OV4gpoasigOR ( $rfcU, $periodoU );
15. OV4gpoasigEX ( $rfcU, $periodoU );
16. OV4actaOR ( $folioU, $claveU );
17. OV4actaEX ( $folioU, $claveU );
18. OV4listaactaOR ( $folioU, $claveU );
19. OV4listaactaEX ( $folioU, $claveU );
20. OV4alumnoactaOR ( $ncU, $folioU, $claveU );
21. OV4alumnoactaEX ( $ncU, $folioU, $claveU );
22. OV4actacalificadaOR ( $folioU, $claveU );
23. OV4actacalificadaEX ( $folioU, $claveU );
```

Del listado anterior, casi todas las funciones son parametrizadas de tal forma que al reducirlas puedan realizar la misma acción sin alterar la estructura del sistema. Por ejemplo, tienen dos características en común:

1. Verifican que los argumentos estén completos, es decir, que los parámetros que reciben sean válidos. Por ejemplo, si la función es *OV1Usuario()* y recibe como argumentos la variable *$ncU* y *$pwU*, dentro de la función tiene la siguiente instrucción.

```
if ($ncU=='' || $pwU=='') return false;
```

Cabe observar que a la construcción del *if* le faltan sus llaves y que pasa exactamente igual para las funciones que reciben tres argumentos como parámetros, de donde verifica cada variable que en caso de ser una cadena vacía, devuelve el valor de falso y esto evita realizar una consulta innecesaria de donde se obtiene el mismo resultado.

2. En caso de no haber error al pasar los argumentos, continúa con la ejecución de la función:

    - En todas las funciones se realiza una consulta de cláusula *SELECT* de SQL para escoger columnas y valores derivadas de éstas.
    - En la parte de la condición se incluye a los argumentos que recibe.
    - El nombre de la tabla en algunas ocasiones es diferente.
    - Casi la mitad de las consultas, piden todas las columnas de la tabla, uso del carácter asterisco: '*'.

Salvo estas diferencias, la estructura de las funciones es exactamente la misma bajo pequeños cambios que muestro subrayado en mayúsculas e indico la acción que realiza dentro del código. Se ilustran las agrupaciones de los códigos de las funciones tal y como se encontraron.

A. Las funciones *esUsuario()*, *esProfesor()* y *OV1Usuario()*, reciben dos argumentos que corresponden a su usuario y contraseña:

```
function NOMBRE DE LA FUNCION ( $arg1, $arg2 ) {
   // Verifica que estén los dos campos completos.
   if ($arg1=='' || $arg2 =='') return false;
   // Búsqueda de los datos de usuarios para iniciar sesión.
   $dbConn = conectar();
   $query = "SELECT COLUMNAS FROM TABLA WHERE CONDICION PARA $arg1";
   $resultado = mysql_query ($query, $dbConn);
   $row = mysql_fetch_array ($resultado);
   $password_from_db = $row ['pw'];
   mysql_close($dbConn);
   // Verifica que el pass enviado sea igual al pass de la db.
   if ( $password_from_db == $arg2 ) {
       return $row;
   } else return false;
}
```

B. Las funciones *OV4listaactaOR()* y *OV4listaactaEX()* reciben dos argumentos que corresponde al número de folio del acta (para calificar o consultar) y la clave de la asignatura para ver la lista de los alumnos que ha calificado o va a calificar.

```
function NOMBRE DE LA FUNCION( $folioU, $claveU ) {

     // verifica que esten los dos campos completos.
     if ($folioU=='' || $claveU=='') return false;

     // busqueda de los datos de usuarios para iniciar sesión.
    //  $query = "SELECT idUsuario, usuario, password, tipo FROM
`usuarios` WHERE usuario = '$usuario'";
     $consultota=array();
     $countU=0;

     $dbConn = conectar();
     $query = "SELECT * FROM TABLA WHERE MISMA CONDICION PARA AMBOS
ARGUMENTOS ORDER BY COINCIDEN EN LA COLUMNA DE ORDEN ";
     $resultado = mysql_query ($query, $dbConn);

     while ($row = mysql_fetch_array ($resultado, MYSQL_ASSOC))
     {
          $consultota[$countU]=$row;
          $countU++;
     }

     mysql_close($dbConn);
//   if ( $password_from_db == md5($pwU) )
    {
          if ( $countU> 0 )
          {
          return $consultota;
          } else return false;
     }
}
```

C. Las funciones *OV4actacalificadaOR()* y *OV4actacalificadaEX()* reciben dos argumentos que corresponden al número de folio del acta y la clave de la asignatura para verificar si todos los alumnos han sido calificados.

```
function NOMBRE DE LA FUNCION( $folioU, $claveU ) {

     // verifica que esten los dos campos completos.
     if ($folioU=='' || $claveU=='') return false;

     // devuelve true si no encuentra calificaciones nulas

     $consultota=array();
     $countU=0;
     $bandera=true;

     $dbConn = conectar();
     $query = "SELECT * FROM TABLA WHERE MISMA CONDICION PARA AMBOS
ARGUMENTOS ORDER BY COINCIDEN EN LA COLUMNA DE ORDEN";

     $resultado = mysql_query ($query, $dbConn);

     while ($row = mysql_fetch_array ($resultado,MYSQL_ASSOC))
     {
          $consultota[$countU]=$row;
          $countU++;
          if ($row['calificacion'] == NULL OR $row['calificacion']
== "" ) $bandera=false;

     }

     mysql_close($dbConn);

     if ( $bandera )
     {
     return true;
     } else return false;

}
```

A pesar de esto, la estructura de cada agrupación sigue coincidiendo salvo en pequeñas diferencias que pueden ser unificadas. Básicamente, todas las funciones realizan una solicitud de información: verifican los parámetros, realiza la consulta a la base de datos, verifican si hay o no registros para después devolver el listado o devolver el valor lógico de falso.

Siendo un sistema de evaluación, el personal académico tiene la posibilidad de calificar o cerrar el acta de la asignatura correspondiente; esto quiere decir que hay un archivo que realiza las modificaciones necesarias en la base de datos de acuerdo a lo que el profesor realice, parte del programa se muestra en el **Código 3**Código 1 que expongo en el **Anexo**.

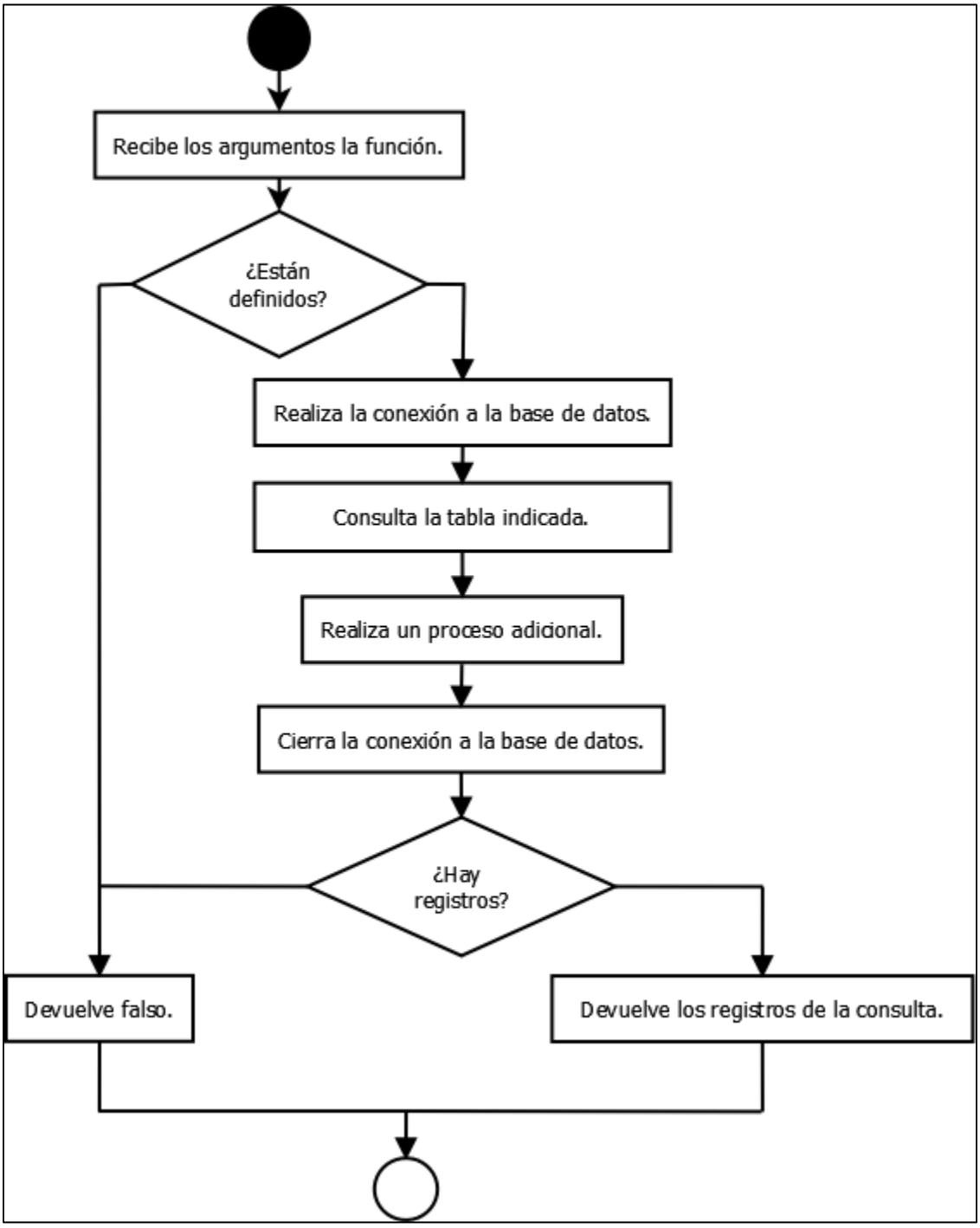


Figura 8. Comportamiento general de las 23 funciones.

Una vez generada la consulta con los datos seleccionados por el profesor, pasa a ejecutar la función *ejecutaConsulta()*, en la cual se reitera la estructura de las otras funciones salvo que el argumento a verificar es la construcción de la consulta.

```
function ejecutaConsulta ( $operacion ) {
     // Verifica que estén los dos campos completos.
     if ($operacion=='' ) return false;

     // Búsqueda de los datos de usuarios para iniciar sesión.
     $dbConn = conectar();
     $result = mysql_query($operacion, $dbConn);
     mysql_close($dbConn);

     if ( $result==true ) {
          return $result;
     } else return false;
}
```

La estructura de la función anterior, me ayudó a crear mi propuesta para unificar las funciones que repiten el comportamiento de ingresar y ejecutar la consulta para luego procesarla de acuerdo a los detalles que tienen de diferencia como devolver la lista, verificar la existencia de datos o comprobar determinada información.

De la misma forma, para evitar la repetición del código y centralizar todas las funciones en un solo archivo al que puedan acceder cualquiera de los sistemas implementados o los que necesiten ser creados, elaboré las funciones: *muestraError()*, *consultaAArreglo()* y *consulta()*.

El objetivo de la función *muestraError()* que se encuentra en el **Anexo** como **Código 4**, es mostrar el mensaje de error en caso de que haya ocurrido algún problema con el resultado de la consulta que se haya solicitado a MySQL. En donde el primer parámetro (la variable *$resultadoConsulta*) almacena el conjunto de resultados de MySQL en caso de éxito, o *FALSE* en caso de haber algún error. El segundo parámetro (la variable *$activaMensaje* definida con el valor de *FALSE* por defecto), indica el tipo de mensaje que debe ser mostrado al usuario que acceda al sistema, para este caso si es el personal académico, es un mensaje de error generalizado (indicando que ha ocurrido un problema y que se recomienda que su solicitud se haga más tarde); y para verificar la existencia del error o realizar ciertas pruebas, puede mostrar el error explícito del problema que tuvo con determinada solicitud (siempre y cuando la variable *$activaMensaje* se defina con el valor de *TRUE*). Por último, si no existiera error alguno, la función no realiza ninguna acción y la ejecución del programa continúa sin ningún problema.

La función *consultaAArreglo()* que se encuentra en el **Anexo** como **Código 5**, se activa en el caso del resultado de una consulta de tipo *SELECT*; lo que hace es convertir el conjunto de resultados MySQL (de la variable *$resultado*) en un arreglo indexado, de tipo numérico, si la variable *$tipoArreglo* es igual a "n", un arreglo asociativo (donde las claves son caracteres) si la variable *$tipoArreglo* es igual a "a" o si se requieren ambos (si la variable *$tipoArreglo* es igual a "b" o si no fue definida); en caso de que no haya registros, simplemente se devuelve la variable *$arreglo* como un arreglo vacío.

Por último, la función *consulta()* que se encuentra en el **Anexo** como **Código 6**, es la encargada de utilizar las funciones anteriormente mencionadas y de englobar cuatro tipos de consulta en MySQL: *UPDATE*, *INSERT*, *DELETE* y *SELECT*. Además de unificar las funciones repetidas, tiene la característica de que puede ser utilizada en cualquier sistema que use o necesite este tipo de consultas, siempre que el tipo de consulta seleccionado siga con las siguientes características:

1. **Tipo UPDATE**:

    - La variable *$tipo* debe contener el valor de 'a'.
    - La variable *$tabla* contiene el nombre de la tabla.
    - La variable *$columnas* debe contener la sintaxis de las columnas a modificar en MySQL.
    - La variable *$condicion* puede contener alguna condición al momento de hacer la modificación, o bien, se realizará por defecto en todos los registros si la variable no está definida.

2. **Tipo INSERT**:

    - La variable *$tipo* debe contener el valor de 'i'.
    - La variable *$tabla* contiene el nombre de la tabla.
    - La variable *$columnas* debe ser un arreglo asociativo a manera que las claves corresponden al nombre de las columnas de la tabla especificada con los respectivos valores a insertar en el registro.
    - La variable *$condicion* puede contener alguna condición al momento de hacer el registro, o bien, se realizará por defecto la inserción del registro si la variable no está definida.

3. **Tipo DELETE**:

    - La variable *$tipo* debe contener el valor de 'b'.
    - La variable *$tabla* contiene el nombre de la tabla.
    - La variable *$columnas* debe contener alguna condición que le indique la eliminación de los registros de cierta tabla, o bien, se realizará por defecto la eliminación de todos los registros si la variable no está definida.

4. **Tipo SELECT**:

    - La variable *$tipo* debe contener el valor de 's'.
    - La variable *$tabla* contiene el nombre de la tabla.
    - La variable *$columnas* puede contener el nombre de las columnas seleccionadas y que estén definidas en la tabla a la que se solicita la información, o bien, se solicitarán todas las columnas, por omisión, mediante el carácter asterisco (*) cuando la variable no está definida.
    - La variable *$tipoArreglo* puede contener el tipo de arreglo seleccionado en caso de que haya registros existentes, ya sea un arreglo indexado o numérico (con el valor igual a 'n'), asociativo (que tiene con el valor por defecto igual a 'a' en caso de no estar definida) o ambos (con el valor igual a 'b').

Cuando la función se ejecuta, el primer parámetro (la variable *$tipo*) indica qué tipo de instrucción se trata en la sentencia *switch*, realiza la debida construcción de su sintaxis para crear el enlace de la conexión a la base de datos correspondiente y ejecutar la instrucción mediante dicha conexión; se verifica si durante la ejecución ocurrió algún error, o bien, en caso de éxito obtener un arreglo de los registros obtenidos si la consulta es de tipo *SELECT* o en caso contrario, devolver el valor de *TRUE* para los demás tipos de consultas (*UPDATE*, *INSERT* o *DELETE*).

Esta función logra sintetizar las 23 funciones que anteriormente listé, y que a su vez pueden sustituir a otras funciones con el mismo comportamiento o que realicen las consultas de tipo *INSERT* y *DELETE* que incluí para unificar lo mejor posible a las solicitudes en MySQL, tanto para el uso del sistema EVA, como los sistemas en funcionamiento existentes o los que deba crear y que requieran el uso de estas ejecuciones. Este cambio genera una mejora en el tiempo de respuesta y la carga del archivo en cuanto al tamaño y al número de funciones que deban precargarse.

Un problema similar en importancia, es en cuanto a la vista o modificación de estos datos al usuario; para ello, el sistema requiere como apoyo un archivo en el que se encuentran las siguientes funciones:

```
1. function do_html_URL($urlU, $nameU)
2. function muestra_inscritas($cat_array)
3. function tabla_asignaturas($reg_array)
4. function ordtabla_asignaturas($reg_array)
5. function ordtsimple_asignaturas($reg_array)
6. function tabla_cardex($reg_array)
7. function tabla_inscritas($reg_array)
8. function cambiotabla_inscritas($reg_array)
9. function bajatabla_inscritas($reg_array)
10.  function bajamuestra_registro($reg_array)
11.  function tabla_conrepertorio($reg_array)
12.  function tabla_grupos($reg_array)
13.  function ordtabla_grupos($reg_array)
14.  function ordcambiotabla_grupos($reg_array)
15.  function ordtablasimple_grupos($reg_array)
16.  function tabla_gpoasigOR($reg_array)
17.  function tabla_gpoasigEX($reg_array)
18.  function muestra_grupo($reg_array)
19.  function muestramensaje($reg_array)
20.  function tabla_listaactaOR($reg_array)
21.  function tabla_listaactaEX($reg_array)
22.  function tabla_listasimpleactaEX($reg_array)
23.  function tabla_listasimpleactaOR($reg_array)
24.  function display_categories($cat_array)
```

Básicamente este listado sirve de apoyo para la muestra de información que el usuario haya solicitado; sin embargo, ocupa un espacio de tiempo y memoria innecesario, ya que de las 24 funciones que contiene el archivo, la 1, 16, 17, 20 y 21 son las únicas funciones que utiliza el sistema EVA y las otras son código sin usar que ha sido copiado de algún otro lado.

Exceptuando la función 1 y la función 24, las demás funciones tienen la característica de crear una tabla en HTML a partir de un arreglo de registros proveniente de alguna consulta de tipo *SELECT*; para que el arreglo pueda ser mostrado sin problema, debe estar definido al menos con las claves que indica la función a utilizar, de lo contrario arrojarían un error en tiempo de ejecución. Al continuar con esta mecánica, obligaría a memorizar al menos 22 funciones listadas y conocer cada una de las estructuras específicas de cada arreglo que debe tener en cada función, esto es muy ineficiente ya que por cada nueva tabla que necesite mostrar, obliga al desarrollador a generar una nueva función nueva.

Para evitar esto, analicé cada una de las funciones para encontrar un patrón, el cual me permitió generar dos principales estructuras con una ligera diferencia, que es el uso de la función *do_html_URL()* que se encuentra en el **Anexo** como **Código 7**. Ésta genera un hipervínculo con el enlace que contiene la variable *$urlU* bajo el nombre que contiene la variable *$nameU*. Siendo una función necesaria para el sistema EVA para indicar al profesor qué listado de exámenes solicita evaluar o consultar (ya sea ordinario o extraordinario), así como elegir al alumno que va a evaluar en caso de que el profesor no haya calificado y cerrado el acta (es decir, confirmar sus evaluaciones). De las 22 funciones que van de la función 2 a la función 23 de la lista, tiene dos estructuras que generalizan su comportamiento.

La primera estructura (**Código 8** en el **Anexo**) verifica si el parámetro es una variable de tipo arreglo, en caso de que no lo sea, entonces envía un mensaje indicando que no hay registros; en caso de ser un arreglo, crea una tabla con N columnas y mediante el ciclo *foreach* muestra cada uno de los registros existentes del arreglo que va a ser mostrado, construye el hipervínculo en alguna de las columnas siempre y cuando el arreglo esté construido bajo las necesidades de la función *do_html_URL()*, ya que de lo contrario arrojaría un error.

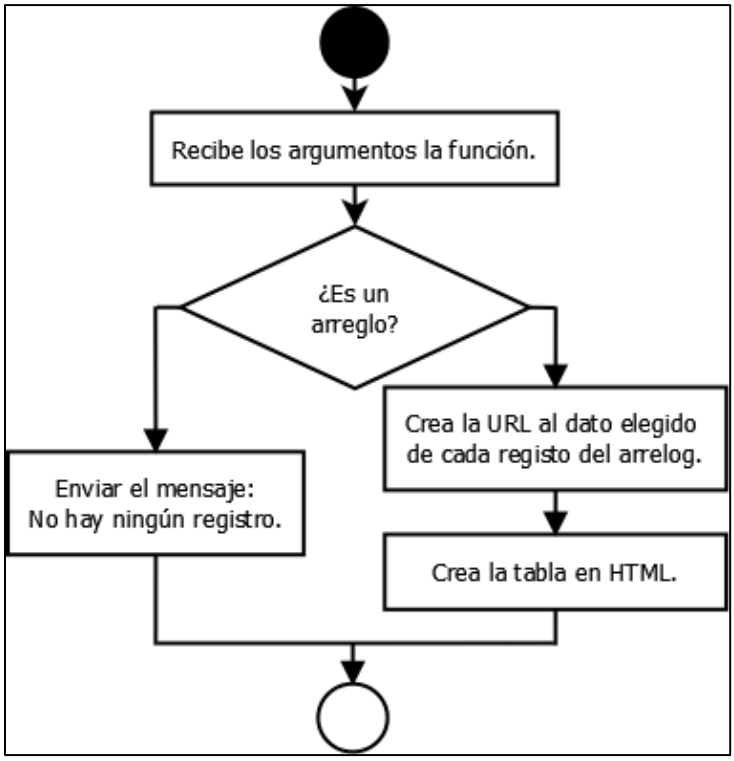


Figura 9. Creación de una tabla con hipervínculos.

La segunda estructura (**Código 9** en el **Anexo**) tiene el mismo comportamiento que la anterior, la única diferencia es que no crea ningún hipervínculo, sólo crea la estructura de una tabla en HTML con todos los registros del arreglo que contenga el parámetro *$reg_array*.

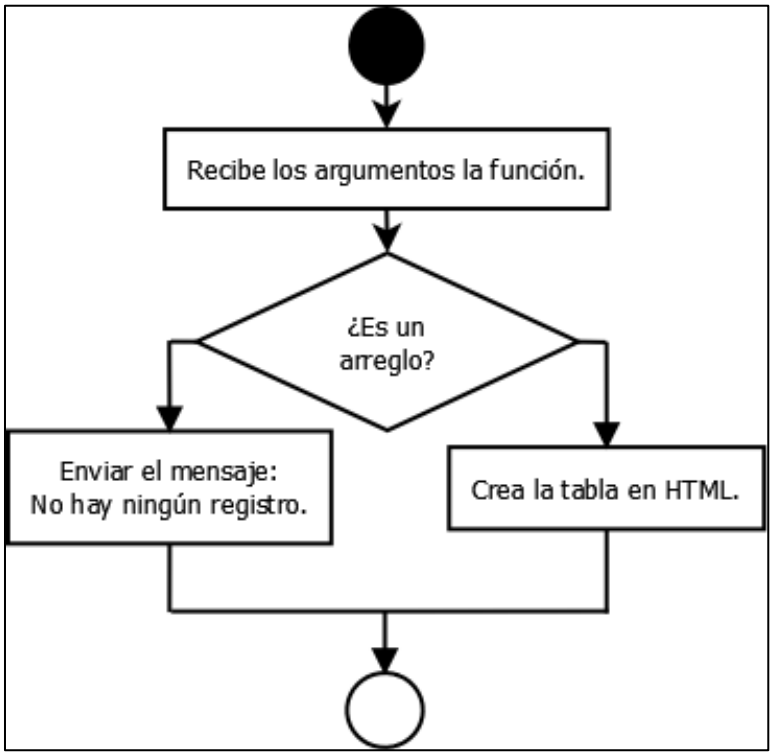


Figura 10. Creación de una tabla sin hipervínculos.

En esencia, ambas estructuras hacen exactamente lo mismo: verifican el parámetro o crean la tabla en HTML a partir de los valores establecidos dentro de la función. Al dejar aparte la creación de hipervínculos como una función adicional, fue posible crear una única función que realiza la misma ejecución para ambas estructuras, para esta unificación me apoyé de una función de PHP llamada *implode()*, que devuelve una cadena que contiene la unión de todos los elementos de un arreglo (asociativo, número o ambos) respetando su orden, dependiendo de la cadena que determine la unión de los elementos. A modo de ejemplo puede verse en el **Código 10** del **Anexo**.

El comportamiento de la función *implode()*, es la responsable de resolver el problema de querer mostrar un número de columnas indeterminado, siendo el principal conflicto entre las funciones por las cuales existe más de una; gracias a que la función puede recibir cualquier arreglo de N elementos y devuelve la cadena compuesta por los mismos N elementos de acuerdo a la cadena de unión elegida. De esta forma, si la cadena de unión indicó que fuera la etiqueta HTML de cierre de una celda en una tabla y la de apertura de una celda ('</td><td>'), entonces obtengo lo siguiente:

```
<?php

    $arreglo = ['Columna 1', 'Columna 2', ..., 'Columna i', ...,
'Columna N'];

    // Se une cada elemento por las etiquetas HTML '</td><td>'.
    echo implode('</td><td>', $arreglo);

?>
```

El resultado es la siguiente cadena:

Columna 1</td><td>Columna 2</td><td>...</td><td>Columna i</td><td>...</td><td>Columna N

Si a este resultado se le antepone la etiqueta HTML de apertura de una celda (<td>) y al final el de la etiqueta HTML de cierre de un celda (</td>), se construye la sintaxis correcta para generar N celdas en HTML.

<table border = 1>

<td> Columna 1 </td>
<td> Columna 2 </td>
<td> ... </td>
<td> Columna i </td>
<td> ... </td>
<td> Columna N </td>

</table>

Una vez que se incluyen las N celdas dentro de las etiquetas HTML para crear una tabla (*<table>* y *</table>*), obtengo la siguiente vista:

Columna 1	Columna 2	...	Columna i	...	Columna N

Gracias a este resultado, logré resolver la construcción de una tabla de cualquier arreglo sin importar cuantas columnas contenga, haciendo que la función propuesta se volviera el sustituto de las 22 funciones de la lista y principalmente, sustituir las cuatro funciones que utiliza el sistema EVA. La solución está dada en el **Código 11** del **Anexo**, ésta función llamada *muestraTabla()* recibe dos argumentos, el primer argumento debe ser un arreglo asociativo o número que contiene los registros de alguna consulta de tipo *SELECT*; el segundo es un arreglo opcional que debe contener los valores de los nombres como quieran que se llamen cada una de las columnas de la tabla; si la variable *$encabezado* no es enviada, entonces se omitirá para mostrar únicamente los registros de la tabla. En caso de que la variable *$registros* no sea un arreglo, se lanza el mensaje "No hay registros" tal como las otras funciones lo tenían.

Por último, para integrar el comportamiento de incluir el hipervínculo para la necesidad de algunas tablas, elaboré una función adicional para no mezclar comportamientos que no son necesarios. Retomé y modifiqué la función *do_html_URL()* para que devuelva la construcción del hipervínculo como una cadena para almacenarla más adelante, se puede ver en el **Anexo** como **Código 12**.

Con esta diferencia, la siguiente función hace el trabajo para creación de los hipervínculos para cualquier arreglo determinado que lo necesite sin alterar la estructura, es decir, únicamente modifica el valor que debe aparecer como hipervínculo. El código de la función se muestra como **Código 13** del **Anexo**.

La función llamada *hipervinculos()*, el primer argumento es el arreglo que será transformado para ser una tabla en HTML, el segundo argumento (la variable *$columna*) es el índice al que se convertirá en un hipervínculo junto con los dos argumentos siguientes, los cuales consisten en indicar la dirección URL al que va a ser dirigida (la variable *$enlace*) y en el caso de enviar datos mediante el método GET, se enviarán como arreglo asociativo en la variable *$variablesPorGet*. Por ejemplo, si tengo un arreglo con M registros y en cada uno de los registros se creará un hipervínculo en la iésima columna:

```
            $arr = array
            (

                ['Renglón 1'] => array(...),
                    ...,
                ['Renglon k-esimo'] => array
                (

                    ['Indice1'] => 'Columna 1',
                    ['Indice2'] => 'Columna 2',
                    [   ...   ] =>     ...,
                    ['Indicei'] => 'Columna i',
                    [   ...   ] =>     ...,
                    ['IndiceN'] => 'Columna N'

                ),
                    ...,
                ['Renglón M'] => array(...)

            );

    $arr = hipervinculos(
       $arr, 'Indicei', 'URL',
       array('Indice1' => 'Columna 1', 'Indice2' => 'Columna 2')
    );
```

La función *hipervinculos()* indica que el arreglo llamado *$arr*, en la columna 'Indicei' será un hipervínculo que dirigirá al usuario a la dirección 'URL' y enviaré los datos que corresponden a las variables de las claves 'Indice1' e 'Indice2' (es conveniente señalar que pueden ser cualquiera de las columnas del arreglo o datos que no sean parte del arreglo). De esta forma, el resultado sería el siguiente:

```
$arr = Array
(

    ['Renglón 1'] => array(...),
        ...,
    ['Renglon k-esimo'] => array
    (

        ['Indice1'] => 'Columna 1',
        ['Indice2'] => 'Columna 2',
        [   ...   ] =>     ...,
        ['Indicei'] => '<a href =
    "URL?Indice1=Columna 1&Indice2=Columna 2">Columna i</a><br />',
        [   ...   ] =>     ...,
        ['IndiceN'] => 'Columna N'
    ),
        ...,
    ['Renglón M'] => array(...)

);
```

Gracias a este cambio, cuando el profesor solicite sus actas de examen ordinario del sistema EVA para evaluar a sus alumnos, el código que tenía originalmente:

```
     $gpoasigOR=OV4gpoasigOR ( $arrUsuario['rfc'], $periodo );
     tabla_gpoasigOR($gpoasigOR);
```

Cambiaría de la siguiente manera:

```
$gpoasigOR = consulta('s', 'actasordinario',
     'folio, clave, asignatura, grupo, calificado ',
     "WHERE rfc = '{$arrUsuario['rfc']}' AND periodo = '$periodo'
ORDER BY asignatura, grupo",
      'a');

$variablesGet = Array('folio' =>'folio', 'clave' =>'clave');
$gpoasigOR = hipervinculos($gpoasigOR, 'folio', 'showactaor.php',
$variablesGet);

$encabezado = array(
    'Folio', 'Clave', 'Asignatura', 'Grupo', 'Calificado'
);

muestraTabla($gpoasigOR, $encabezado);
```

De esta forma, se evita la repetición de código y se centralizan las 22 funciones en una o dos funciones, ya sea para el caso de las funciones que sólo generan una tabla en HTML sencilla o para generar una tabla con hipervínculos.

Por último, durante el cierre de cada periodo semestral, la Secretaría de Servicios y Atención Estudiantil abre el sistema de evaluación con el periodo correspondiente, mediante un duplicado del sistema previo (es decir, el sistema *EVA 2016-1*, se crea a partir del sistema *EVA 2015-2*), realiza una copia de cada uno de los archivos y de la página inicial del sistema, se modifica la variable $periodo por el periodo semestral correspondiente:

```
                $periodo = "2015-2";
                $_SESSION['periodo'] = $periodo;
```

A la variable periodo se le coloca el valor de "2016-1" y se almacena de la misma forma a la variable de sesión *$_SESSION['periodo']*, realizando el resto del comportamiento del sistema, y en el menú principal se crea el nuevo enlace al sistema *EVA 2016-1* para terminar con su creación:

```
    <li>
        <a href = "http://www.enmvirtual.net/eva_161/index.php">
            EVA 2016-1
        </a>
    </li>
```

Para evitar este duplicado de sistemas innecesarios, propuse solucionarlo mediante el envío de parámetros por el método GET. Este método envía los datos por la URL ya sea que hayan sido asignados directamente a la dirección o mediante algún formulario que va a un script de PHP, por ejemplo, si tengo la siguiente dirección:

```
<a href = "http://localhost/pagina.php?arg1=valor1&...&argN=valorN">
    Nombre de la dirección
</a>
```

Este enlace llama al PHP *pagina.php* y le indica que se le pasan una serie de datos. Estos datos se envían al servidor mediante el método GET indicándole con el signo de interrogación de cierre (?), seguido de un número determinado de datos (en este caso N valores) separados por el símbolo de ampersand (&) bajo una estructura de nombre/valor.

Una vez que los datos enviados los obtiene el script de PHP, los convierte en un arreglo asociativo y los almacena en la variable global *$_GET*. De tal manera que para obtener un determinado valor, se realiza lo siguiente:

```
        $arg1 = $_GET['arg1'];
        $arg2 = $_GET['arg2'];
             ...
        $argN = $_GET['argN'];

        echo "Lo valores obtenidos son: ", $arg1, " ",
            $arg2, " ", ..., " ", $argN,".";

        // Tiene como resultado:
        // Lo valores obtenidos son: valor1 valor2 ... valorN.
```

Cabe mencionar que a pesar de que permite introducir una gran cantidad de valores de la forma nombre/valor, el método GET tiene una longitud máxima de 2048 caracteres. De la misma forma no es conveniente usarlo para pasar información que sea sensible como claves, ya que el enlace es visible en el navegador del usuario.

La otra forma es mediante el uso de un formulario HTML, por ejemplo:

```
<form action = "pagina.php" method = "GET">
    Dato1: <input type = "text" name = "arg1" />
    Dato2:
    <select name = "arg2">
        <option value = "1"> Uno </option>
        <option value = "2"> Dos </option>
        <option value = "3"> Tres </option>
    </select >
    <input type = "submit" value = "Enviar" />
</form>
```

Al enviar los datos por el formulario, el enlace que se indica en el atributo *action* de la etiqueta *form*, se le añade cada uno de los datos ingresados y el nombre asignado es el valor que contiene el atributo *name* de cada etiqueta, si en Dato1 el usuario escribiera "Hola" y de la etiqueta *select* en el Dato2 eligiera la opción Tres, el resultado del enlace final lo dirige a:

```
pagina.php?arg1=Hola&arg2=3
```

Así los valores se obtienen de la misma forma mediante el uso de la variable global *$_GET*. Con esto, al usar la primera opción del método GET, de todos los enlaces que van a cada uno de los sistemas de evaluación creados y que se permiten calificar, los redirijo a una única URL y envío como valor el periodo que les corresponde, pasando de estos enlaces:

```
<li><a href="http://www.enmvirtual.net/eva_131/index.php">EVA 2013-1</a></li>
<li><a href="http://www.enmvirtual.net/eva_132/index.php">EVA 2013-2</a></li>
<li><a href="http://www.enmvirtual.net/eva_141/index.php">EVA 2014-1</a></li>
<li><a href="http://www.enmvirtual.net/eva_142/index.php">EVA 2014-2</a></li>
<li><a href="http://www.enmvirtual.net/eva_151/index.php">EVA 2015-1</a></li>
<li><a href="http://www.enmvirtual.net/eva_152/index.php">EVA 2015-2</a></li>
```

A estos:

```
<li><a href="http://www.enmvirtual.net/valida.php?p=2013-1">EVA 2013-1</a></li>
<li><a href="http://www.enmvirtual.net/valida.php?p=2013-2">EVA 2013-2</a></li>
<li><a href="http://www.enmvirtual.net/valida.php?p=2014-1">EVA 2014-1</a></li>
<li><a href="http://www.enmvirtual.net/valida.php?p=2014-2">EVA 2014-2</a></li>
<li><a href="http://www.enmvirtual.net/valida.php?p=2015-1">EVA 2015-1</a></li>
<li><a href="http://www.enmvirtual.net/valida.php?p=2015-2">EVA 2015-2</a></li>
```

Envío el valor del periodo al PHP *valida.php* para realizar la verificación de este valor, como mecanismo de seguridad para evitar que puedan ingresar a otros periodos o quieran romper la seguridad al realizar alguna inyección de código.

La función *getSemestre()* que aparece como **Código 14** en el **Anexo**, calcula el periodo semestral actual de acuerdo a la fecha que se calcule en ese momento, o bien, si se le pasa como argumento un valor entero N regresa el enésimo periodo correspondiente del actual; por ejemplo, si estuviéramos en el semestre 2016-1, la función devuelve como valor "2016-1" si se ejecuta *getSemestre()* o el valor N es igual a cero, es decir, *getSemestre(0)*; ahora, si ejecutamos *getSemestre(-4)*, la función nos dará el periodo "2014-1", ya que retrocede cuatro semestres al actual, que en este caso se calcula como "2016-1".

La función arregloPeriodos*()* que aparece como **Código 15** en el **Anexo**, crea el arreglo de N periodos que se requieran al pasarle un valor mayor a cero en la variable *$total*, de tal manera que si ejecuto *arregloPeriodos(2)* con la misma fecha que en el ejemplo anterior, obtengo el arreglo con las cadenas '2015-1', '2015-2' y '2016-1'. Esta función evita que se acceda a otros periodos que sean inválidos y, más importante aún, evita la inyección de código que puede haber mediante el siguiente procedimiento en *valida.php*.

```
if ( isset($_GET['p']) ) {

    $periodosValidos = arregloPeriodos(5);

    if(in_array($_GET['p'], $periodosValidos) ){

        $_SESSION['periodo'] = $_GET['p'];

    } else{

        $_SESSION['periodo'] = getSemestre();

    }

} else{

    $_SESSION['periodo'] = getSemestre();

}
```

Primero, si el valor de p está definido por el método GET, entonces ejecuto la función *arregloPeriodos(5)* para calcular los seis periodos permitidos para poder calificar y lo almaceno en la variable *$periodosValidos* como un arreglo; después pregunto si el valor que contiene la variable global *$_GET['p']* se encuentra en el arreglo, si es así, entonces lo almaceno en la variable de sesión *$_SESSION['periodo']*, en caso contrario, le asigno el semestre actual por defecto. De la misma forma le asigno el semestre actual en caso de que la variable global *$_GET['p']* no se encuentre definida.

Una vez que termino de definir el periodo, redirecciono a la página principal del sistema de evaluación para que el personal docente inicie sesión y pueda realizar el procedimiento de evaluación de sus alumnos; este sistema, realiza exactamente la misma funcionalidad que cada uno de los sistemas que fueron creados anteriormente (*EVA 2013-1*, *EVA 2013-2*, *EVA 2014-1*, *EVA 2014-2*, etcétera), mostrando las asignaturas a calificar de acuerdo al periodo que el profesor haya elegido, esto es mediante el cambio de la variable *$periodo* que anteriormente señalé:

```
                $periodo = "2015-2";
                $_SESSION['periodo'] = $periodo;
```

Por lo siguiente:

```
    // Si la variable $_SESSION['periodo'] está definida
    if ( !isset($_SESSION['periodo']) ) {

        // Envío al usuario a la página principal.
        header(Location: http://escolares.enmusica.unam.mx/);
        die;

    }
```

Este cambio me permitió verificar que si la variable de sesión *$_SESSION['periodo']* no está definida, los profesores son redireccionados al menú principal de la página donde deben seleccionar nuevamente el enlace del periodo a evaluar, en caso de que estuviera definida, entonces el usuario podrá iniciar sesión y realizar el proceso de evaluación.

De esta manera al darle una estructura más coherente al código, evité la repetición constante de archivos y el gasto de tiempo innecesario que conlleva realizar todo el procedimiento, logrando tener un único sistema de evaluación *EVA* y que sin importar en el periodo en el que nos encontremos, realice el mismo comportamiento que los sistemas anteriores dependiendo del semestre que resulte de la función *getSemestre()* y *arregloPeriodos()* de acuerdo a la fecha que se le haya indicado.

De esta manera al darle una estructura más coherente al código, sus modificaciones son parecidas al patrón MVC (Modelo-Vista-Controlador) para evitar la repetición constante de archivos y el gasto de tiempo innecesario que conlleva realizar todo el procedimiento, logrando tener un único sistema de evaluación *EVA* y que sin importar en el periodo en el que nos encontremos, realice el mismo comportamiento que los sistemas anteriores dependiendo del semestre que resulte de la función *getSemestre()* y *arregloPeriodos()* de acuerdo a la fecha que se le haya indicado.

CAPÍTULO IV
BUENAS PRÁCTICAS DE PROGRAMACIÓN

PHP (acrónimo recursivo de PHP: Hypertext Preprocessor), es uno de los lenguajes de programación que tiene una altísima presencia en internet, esto ha propiciado que entre la comunidad de programadores haya dos vertientes básicas: detractores y defensores. Esta clasificación cae en tela de juicio cuando dos o más individuos dan sus diversos argumentos para entrar en una discusión para temas como: lenguaje interpretado contra compilado, la nomenclatura de funciones, la rapidez, etcétera. A pesar de que en la realidad, LAMP (Linux, Apache, MySQL/MariaDB y Perl/PHP/Python) está detrás de la mayoría de los servicios y las aplicaciones Web.

PHP es un lenguaje de programación fácil de aprender, usarlo de forma experta requiere de tiempo, esfuerzo y dedicación; al ser un código libre, el desarrollo de las aplicaciones deben volverse muy robustas, debe contar con un funcionamiento y una vista lo más adecuada posible, ya que sus problemas pueden estar en todas partes, incluso de las más inesperadas. Estos diversos fallos o vicios, los cometemos a lo largo del tiempo ya sea por la falta de conocimiento, carencia de la experiencia o por la 'rapidez' de entregar el desarrollo de la aplicación y que como consecuencia empeora conforme pasa el tiempo.

Durante mi primer año como trabajador en la Facultad de Música, analicé las diversas aplicaciones que fueron desarrolladas para la "Ventanilla Virtual", encontrando diversas irregularidades conforme fui aprendiendo a programar con el lenguaje de PHP. Observé los errores más comunes como es la falta de documentación, código innecesario, hasta mejorar el funcionamiento del propio lenguaje y volverlo más consistente, mejorando su lectura y la colaboración de otros programadores.

El siguiente listado es parte de algunos estándares más comunes que fui encontrando para la buena práctica de programación usando este lenguaje, lo cual nos permite, entre otras cosas, ser más objetivos a la hora de analizar código ajeno, la mejora de la reutilización de nuestro propio código, la compatibilidad que se debe tener con otros proyectos siendo lo más efectivo posible, entre otros.

Manejo correcto de la etiqueta.

PHP es un lenguaje de código abierto, muy conocido, especialmente adecuado para el desarrollo Web con la posibilidad de ser incrustado en HTML. Para indicar el uso y la ejecución de este lenguaje, se realiza mediante las diversas formas de indicación de las etiquetas o *tags* de apertura y cierre que permiten entrar y salir del "modo PHP". Las variantes para indicar este código, en PHP 5 existen hasta cinco pares de etiquetas que clasifico en tres tipos:

Etiqueta estándar:	`<?php echo "¡Hola mundo!"; ?>`

Etiqueta desde HTML:	`<script language = "php">`
	`  echo "¡Hola mundo!";`
	`</script>`

Etiquetas cortas:	`<? echo "¡Hola mundo!"; ?>`, `<?= "¡Hola mundo!"; ?>` y `<% echo "¡Hola mundo!"; %>`

Tanto la etiqueta estándar *(<?php ?>*) como la que se usa desde HTML (*<script language = "php"> </script>*), siempre estarán disponibles y su etiqueta corta (*<?= ?>*) que es la abreviatura del constructor del lenguaje *echo*, siempre está disponible en PHP 5.4.0 y sus posteriores, siendo así equivalente a (*<?php echo 'imprimir esta cadena' ?>*). Esto indica que dependiendo del entorno en el que la aplicación se encuentre alojada, estas cinco etiquetas pueden crear problemas. Por ejemplo:

`<? ?>`	Está reservado para declarar XML.
`<?= ?>`	Esto no es XML válido (en cambio *<?php ?>* sí lo es).
`<% %>`	Esto es para el uso de ASP (*Active Server Pages*).

Aunque para algunos pueden ser prácticas, las etiquetas abreviadas y las etiquetas al estilo de ASP son menos portables, siendo generalmente no muy recomendadas. La etiqueta corta (*<? ?>*) únicamente funciona si la directiva *short_open_tag* está habilitada en la configuración de PHP de nuestro servidor (archivo *php.ini*), permitiendo que el script se ejecute exactamente igual y sin ningún problema. Pero el problema surge si algún servidor IIS (Internet Information Services) se cambia por otro, dando como consecuencia que las páginas dejarán de funcionar como se espera y, lo que llega a ser más grave, es que puede resultar en que todo el código fuente se imprima por pantalla como texto plano (al no ser interpretado como se espera), quedando a disposición de cualquier usuario.

Así que, la mejor solución para evitarse problemas es mediante la forma estándar (*<?php ?>*), incluso todas las etiquetas deberían ser cambiadas a esta única. Así, se puede garantizar que tenga soporte en el futuro, representando una instrucción de procesamiento válida y compatible si se quiere proveer código PHP a los documentos XHTML o XML, convirtiéndose en una etiqueta única para sus diferentes versiones, debido a que en PHP 7.0.0 algunas etiquetas fueron eliminadas o deben ser habilitadas mediante alguna directiva del fichero de configuración *php.ini*.

Manejo de los tipos de datos.

PHP admite diferentes tipos de datos a pesar de no ser muy estricto, dependiendo de cómo se desarrolle el código esto puede ser tanto una ventaja como una desventaja, dando resultados no esperados. Por ejemplo, si se tiene el siguiente código:

```
<?php

    $var1 = "1";
    $var2 = $var1 + 1;

    // Muestra: string(1) "1" int(2)
    echo var_dump($var1, $var2);

?>
```

Al ejecutar el programa, como resultado obtendríamos que *$var1* es una variable de tipo *string* de tamaño 1 y que almacena la cadena "1" y que $var2 es una variable de tipo entero que guarda el valor numérico 2.

De la misma forma, como los operadores no tienen esta restricción va a ocurrir algo similar en lo siguiente:

```php
<?php

    $var1 = 0;
    $var2 = "0";
    $var3 = false;

    var_dump($var1 == $var2); // 0 == "0" dará true

    var_dump($var1 == $var3); // 0 == false dará true

    // Incluso esto es aún más confuso.
    var_dump( '1' == '1. '); // Resultado: bool(true).

?>
```

Si se hiciera uso de una función sin tomar en cuenta tal situación, el desarrollador no obtendría los resultados esperados. Algunas funciones que retornan un número entero, para PHP se considera como valor booleano de *true*, exceptuando el cero que es tomado como *false*; y esto se debe tener en cuenta a la hora de evaluar ciertas condiciones. Por ejemplo, para la función *strpos()* busca la posición de la primera ocurrencia de una subcadena dentro de otra:

```php
<?php

    $cadena = '¡Hola mundo!';
    $busca = 'l';
    $posicion = strpos($busca, $cadena); // $posicion = 3

    $busca = 'z';
    $posicion = strpos($busca, $cadena); // $posicion = false

    $busca = '¡'; // presente en $cadena
    $posicion = strpos($busca, $cadena); // $posicion = 0

?>
```

Por lo tanto, si se realiza la comparación *$posicion == false*, se tiene el problema de que para los casos en que se busca la presencia de los carácteres 'z' y '¡' sean correctas, esto no es verdad. Para ello, la solución efectiva es saber manejar la comparación por valor (==), o bien, comparar por el tipo y su valor (===), ya que en PHP las variables no tienen un tipo asignado y éste puede cambiar en cualquier momento del tiempo de ejecución, conviene tener en cuenta cómo serán evaluadas las comparaciones en PHP.

Por si fuera poco, ocurre lo mismo cuando se desea generar funciones propias. Para tal efecto, lo más conveniente es asegurar qué tipo de datos se están usando y qué se pretende devolver, en caso de que la función lo necesite. De esta manera con estos cambios se tiene el ejemplo siguiente:

```php
<?php

    $var1 = (int) '1';
    $var2 = $var1 + 1;

    var_dump($var1, $var2); // $var1 y $var2 son enteros.

    $var1 = 0;
    $var2 = "0";
    $var3 = false;

    var_dump($var1 === $ var2);
    // 0 == "0" && (integer) == (string) dará false

    var_dump($var1 === $ var3);
    // 0 == false && (integer) == (bool) dará false

?>
```

Así se garantiza que el resultado sea lo esperado al asegurarse la operación o la conversión de los tipos en PHP y evitar problemas en el futuro.

Manejo de errores.

Una de las ventajas que tiene el lenguaje PHP es que al ser un lenguaje de programación interpretado puede funcionar incluso dando errores, por ello muchos proveedores de hosting tienen deshabilitada la depuración de errores. Siempre que se quiera trabajar en depurar código se debe tener activado de la manera más restrictiva posible, de lo contrario puede resultar contraproducente. Una opción es utilizar la función *error_reporting()* en la que establece cuáles errores de PHP son notificados, al pasarle la constante *E_ALL* como argumento, notifica todos los errores que se produzcan en PHP y obliga al desarrollador a producir código que sea compatible a la versión que tenga el servidor.

Con ello se logra verificar el buen manejo del programa, se tiene que tomar en cuenta todas las advertencias que arroje PHP, saber qué métodos están en desuso, índices inexistentes, etcétera. Por ejemplo, si la versión de PHP fuera 5.0 y se utiliza el nivel de error *E_STRICT*, en el siguiente código:

```php
<?php

    if( is_a( $objeto, 'NombreDeLaClase' ) ){
        $objeto->algunMetodo();
    }

?>
```

Es común que se encuentre el error bajo el uso de la función *is_a()* en lugar del operador *instanceof* para determinar si una variable de PHP es un objeto instanciado de una cierta clase. Entonces una solución sería:

```php
<?php

    if( $objeto instanceof 'NombreDeLaClase') ){
        $objeto->algunMetodo();
    }

?>
```

Otro error muy común, es la posibilidad de ocultar los errores o excepciones en el script con el operador de control de errores: el símbolo arroba (@). Este operador funciona anteponiéndose sólo sobre expresiones tales como variables, llamadas a funciones, al incluir un archivo en específico, operaciones, constantes y así sucesivamente. Usarlo NO resulta recomendable, ya que provoca un costo en el rendimiento por cada vez que se utiliza en determinado programa. Por medio del uso del **Código 16**, **Código 17** y **Código 18** del **Anexo**, al colocar la instrucción "*$define = @$i;*" dentro del ciclo provoca un retraso de ejecución muy alto a diferencia de la instrucción "*$define = $i;*", en la que no pide comprobar algún error.

Tiempo empleado en segundos con "@":	Tiempo empleado en segundos sin "@":
1.7058291435242	0.24588298797607
1.6782519817352	0.24556016921997
1.6533670425415	0.25978899002075
1.6076538562775	0.24504590034485
1.7048571109772	0.24538898468018

Siempre hay que evitar hacer uso de este operador, para ello se pueden recurrir a ubicar (con el bloque *try{ } catch( ){ }*) el error o verificarlo con antelación mediante varias pruebas. La única manera de hacer uso de este operador, es que no haya otra manera o que sea estrictamente necesario.

Manejo de excepciones.

A PHP se le incluyeron excepciones, a pesar de ser bueno puede resultar peligroso, ya que son una gran herramienta para manejar diversas situaciones excepcionales en la ejecución del código, pero tienden a usarse de modo incorrecto e incluso abusar de ellas, al igual de que causan una pérdida de memoria cuando son ejecutadas en algunas situaciones. Un ejemplo de abuso podría ser algo como lo siguiente:

```php
<?php

    function verificaDatoDeEntrada ($var){

        if( $var !== "¡Hola!" ){
            throw new Exception("Uso del dato incorrecto");
        }

    }

?>
```

Cuando lo más correcto puede resultar en:

```
<?php

    function verificaDatoDeEntrada($var){
        if( $var !== "¡Hola!" ){
            return false;
        } else{
            return true;
        }
    }

?>
```

Manejar los errores en un bloque de procesos:

```
<?php

    try{

        $persona->setNombre('Luis');
        $persona->setDireccion('Calle 41');
        $persona->setFechaNacimiento('1999-11-28');
        $persona->guarda();

    } catch (Exception e){

        throw new DataPopulationException(
            'Incapaz de guardar el dato para la persona '
            . $persona->dameId(), e);

    }

?>
```

Por último, para ejemplificar la pérdida de memoria cuando se abusa de las excepciones es cuando se lanzan continuamente como en un ciclo de proceso.

```
<?php

    foreach($i = 0; $i < 1000000; ++$i){
        throw new Exceptions('¡Pierdo memoria!');
    }

?>
```

Aquí la memoria que se pide no se libera por completo cada vez que se ejecuta la excepción.

Uso de un depurador.

En ocasiones se necesita examinar la estructura de un programa y conocer cierta información de las variables, usar la función *print_r()*, en la que muestra todos los valores que contenga la variable solicitada o, si se necesitara información más detallada como a qué tipo correspondiera cada uno de esos valores, la función *var_dump()* haría el trabajo. Aunque como éstas u otras funciones, es recomendable usar un depurador, ya que tiene características que ayudan mucho más y evita que sea complicado analizar ciertos errores que surjan, debido a que existen errores en PHP que no son muy informativos para los casos que se tornan moderadamente complejos. De esta manera el depurador proporciona una mayor información indicando el orden de la ejecución del programa, así como la llamada que cause el error.

Uno de los depuradores más recomendados es Xdebug (http://xdebug.org), debido a que:

- Es código libre.
- Lo usan muchos desarrolladores.
- Se integra a PHP como una extensión.
- Se integra con IDEs.
- Tiene opciones para obtener perfiles de uso y cobertura de código.

Entorno de Desarrollo Integrado (IDE).

Si se piensa realizar una aplicación que lleve tiempo y necesite de una organización de código, se debe utilizar un IDE (*Integrated Development Environments*) es un editor de código que ofrece la posibilidad de depurar, autocompletar e indentar el código, características que deben cubrirse y que son muy importantes para el desarrollo del software. Existen diversas alternativas, gratuitas y de pago. Entre las opciones gratuitas destacan Eclipse y Netbeans; por su parte, Zend Studio y PHP Designer destacan como opciones de pago.

Convenciones.

Un punto importante que se debe tomar en cuenta, es estandarizar el código, esto evita las dificultades de la lectura, lo cual resulta grave en un grupo de desarrollo y ayuda a darle mantenimiento sin problema. Para ello hay ciertas convenciones:

- Los nombres de las clases deben ser *MixedCase* o también denominadas *CamelCase*, con la inicial en mayúscula. Una buena práctica de esto, ayuda a manejar la jerarquía de las clases, al igual que ubicar los archivos de forma jerárquica y ordenada. Ejemplo: *ElNombreDeMiClase*
- Las constantes de código como los generados con *#define* de C/C++ o *define()* en PHP siempre deben ir en *ALL_CAPS* o mayúsculas. Ejemplo: *MI_COLOR*.
- Las variables, funciones, propiedades, métodos y parámetros en *camelCase*. Ejemplo: *$variableQueAlmacenaAlgo* o *nombreDelMetodo()*.
- Los métodos llamados "Accessors" utilizaran los prefijos *set* y *get*, usando la regla anterior, por ejemplo *setVar1* y *getVar1*.

- Propiedades, métodos y variables que no sean públicos, los que sean privados serán precedidos por un guión bajo (underscore-prefixed). Ejemplo: *$_miPalabraSecreta*.
- Para los archivos, sólo caracteres alfanuméricos, comenzando en mayúscula, los espacios no están permitidos.
- Indentación, se recomienda usar sólo espacios y no tabs, cuatro espacios por cada nivel de indentación, la mayoría de los editores pueden ser configurados para convertir esos tabs en espacios ayudando a cumplir el requisito.
- Para métodos y funciones, cuando el parámetro de un método sea opcional, y no se tenga preferencia sobre algún valor por defecto se debe usar el valor *null* en lugar de *false*. También, los parámetros deberán estar separados por un espacio en blanco después de cada coma.

Un ejemplo para ilustrar es el siguiente:

```php
<?php

    class MiClase{

        const LIMITE_DE_ITERACIONES = 10;
        public $miPropiedad;
        public $miOtraPropiedad;

        // Se observa el guión bajo al no ser público
        protected $_miPropiedadProtegida;

        // constructor
        function __construct(){

        }

        public function hacerAlgoConMiClase($parametroRecibido){

            for ($i = 0; $i < self:: LIMITE_DE_ITERACIONES; ++$i){

                // Bloque de cualquier código.

            }

        }

        public function dameMiPropiedadProtegida (){

            return $this->_miPropiedadProtegida;

        }

    }

?>
```

Al mostrar o imprimir el mensaje de una cadena.

Para PHP, tanto el constructor de lenguaje *echo* (el cual no es considerada como una función) como el de *print* realizan la misma acción: mostrar cadenas. Salvo que la diferencia radica en que *echo* es sensiblemente más rápida que *print*.

Concatenación de cadenas.

Para el manejo de cadenas en PHP lo más usual en las que se puede especificar, es en cuanto al uso de comillas simples (') o las comillas dobles ("). La diferencia es que en las comillas dobles permite desplegar el valor de una variable aunque esté dentro de las comillas a diferencia de la comilla simple, por ejemplo:

```
<?php

    $fruta = 'manzana';

    echo 'Me gusta la $fruta.';
    // Muestra: Me gusta la $fruta.

    echo "Me gusta la $fruta.";
    // Muestra: Me gusta la manzana.

?>
```

Aunque resulte una forma más visible la introducción de variables en la cadena, no es tan favorable a la hora de mostrar el mensaje, se recomienda evitar siempre el uso de comillas dobles a menos que sea sumamente necesario. La razón es que PHP analiza el contenido de las comillas dobles en búsqueda de variables que deban ser interpretadas, resultando en un tiempo de ejecución mayor a diferencia de usar las comillas simples y empleando el uso del constructor del lenguaje *echo*. De la misma forma al usar *echo*, se deben concatenar las cadenas por medio de comas y no por medio del punto, ya que requiere de menos tiempo al compilador el uso de comas con *echo*, así es más práctico si se tiene:

```
<?php

    echo 'Hola', $nombre, ', ¿qué te trae por aquí?';

?>
```

A diferencia de

```
<?php

    echo 'Hola' . $nombre . ', ¿qué te trae por aquí?';

?>
```

Entonces el peor caso posible sería escribir:

```
<?php

    print "Hola $nombre, ¿qué te trae por aquí?";

?>
```

Búsqueda de cadenas y sus patrones.

Otra característica en PHP, es que existen funciones en las que al realizar la búsqueda de una cadena o un patrón, se tiene la opción de realizarla con o sin importar mayúsculas o minúsculas. Aunque esto puede ser una ventaja, también puede crear problemas en tiempo de ejecución, ya que la búsqueda sin importar si es mayúscula o minúscula es mucho más lenta que el caso en el que no se ignore tal opción. A su vez, para las expresiones regulares, las búsquedas sensibles como *preg_match("/$patron/", $cadena)* son, como norma, ligeramente más eficaces que su equivalente no sensible: *preg_match("/$patron/i", $cadena)*.

Muy recomendable es que, si las coincidencias se realizan de modo iterativo (dentro de un ciclo *for*, *while* o *foreach*), hay que convertir a minúsculas o mayúsculas antes y realizar las operaciones en su versión sensible.

Evitar calcular un valor dentro de ciclos.

La función *count()* nos devuelve el total de todos los elementos de un arreglo o algo de un objeto. El mal uso implica que, por cada vez que se llama a esta función, aumenta hasta en un 50% el tiempo de ejecución dependiendo del tamaño del arreglo. Esto es que si el arreglo es suficientemente grande, el primer bloque de código resulta ser mucho más lento que el segundo, esto se logra ver nuevamente mediante el uso del **Código 16**, **Código 17** y **Código 18** del **Anexo**, por medio del código siguiente:

```
<?php

    // Uso incorrecto
    for ($i = 0; $i < count($arreglo); $i++){
        // Bloque de código.
    }

    // Uso correcto
    $limite = count($arreglo);

    for ($i = 0; $i < $limite; $i++){
        // Bloque de código.
    }

?>
```

Función	Dentro del ciclo *for*		Fuera del ciclo *for*	
	Promedio de tiempo en segundos	**Memoria utilizada en bytes**	**Promedio de tiempo en segundos**	**Memoria utilizada en bytes**
count()	0.233441655	376	0.050306664	464

Al ejecutar las líneas de código anteriores se tiene como resultado que la forma más correcta es realizar el conteo de los elementos (de la variable *$arreglo*) antes de realizar las iteraciones requeridas a menos que sea estrictamente necesario, ya que si el flujo del programa fuera el mismo, sería más conveniente usar una pequeña cantidad de memoria adicional (por la variable adicional) a cambio de un tiempo eficiente de respuesta.

Operadores de incremento o decremento.

En PHP, este tipo de operador puede afectar el valor numérico o de una cadena de acuerdo al orden requerido, es decir, modificar el valor para utilizarlo después (pre-incremento o pre-decremento) o utilizar primero su valor para después modificarlo (post-incremento o post-decremento). Aunque en esencia tanto el pre-incremento (*++$n*) como el post-incremento (*$n++*) de una variable tiene la misma finalidad, la primera opción resulta un 10% más rápido que la segunda. La razón fundamental es que cuando hay post-incremento, el intérprete de PHP necesita crear una variable temporal en la que almacena el valor que va ser incrementado. Un ejemplo para mostrar la diferencia de tiempos es el siguiente bloque de código con la ayuda del **Código 16** y **Código 17** del **Anexo**:

```
<?php

    // Hago un ciclo para hacer un
    // programa que tarde un poco
    for ($i = 0; $i < 3000000; VARIABLE){

        // Bloque de algún código.

    }

?>
```

El tiempo empleado en segundos si en **VARIABLE** se realiza:

Un pre-incremento (*++$i*) es:	Un post-incremento (*$i++*) es:
0.12536692619324	0.16984295845032
0.12285685539246	0.17230105400085
0.12815594673157	0.14654612541199
0.12462711334229	0.14724707603455
0.12985205650330	0.15019011497498

El resultado es similar para pre-decremento (*--$i*) y post-decremento (*$i--*).

Incluir código.

Para evitar la repetición de un código o un programa completo, se necesita almacenar en uno o varios archivos para ser importados por medio de cuatro funciones: *include*, *require*, *include_once* o *require_once*. Su comportamiento es el siguiente:

include('archivo.extension'). Permite incluir el archivo sin importar las veces que se requiera. La función agrega el archivo siempre y cuando exista, de lo contrario emite una advertencia y continúa con la ejecución del programa.

include_once('archivo.extension'). Es similar a *include* salvo que sólo permite incluir el archivo una única vez (tal como el nombre lo indica).

require('archivo.extension'). Su comportamiento es el mismo que *include*, la diferencia radica en lanzar un error fatal en caso de que el archivo no exista, deteniendo la ejecución del programa.

require_once('archivo.extension'). Similar al comportamiento de *require*, salvo que sólo carga el archivo una única vez.

Aunque la explicación es casi la misma, el tipo y la necesidad del archivo que se va a cargar es lo que puede marcar completamente la diferencia. No es lo mismo cargar parte de un programa en PHP que un simple encabezado en HTML, si la página depende del archivo para su correcta ejecución se recomienda usar *require* o *require_once*, en cambio si no es tan relevante se utiliza *include* o *include_once*.

Ahora, al usar *require* o *include* se tiene el riesgo de redefinir clases, funciones o hasta la reasignación de valores en las variables, para evitar ese problema es mejor modificar el archivo y se llame una o más veces; no se debe usar *require_once* o *include_once* a menos que sea absolutamente necesario, ya que estas sentencias consumen muchos recursos al creer que se está ahorrando la verificación de la inclusión del archivo.

Mediante la ejecución del **Código 19** con el uso del **Código 16**, **Código 17** y **Código 18** del **Anexo**, obtengo los siguientes resultados al elegir el tipo de sentencia usada mediante la variable *$sentencia* e indicar la ruta de mi archivo especificada en la variable *$archivo*, al realizar las 10 000 iteraciones indicadas en el ciclo obtengo lo siguiente:

Sentencia	Archivo de texto plano de 189 bytes		Archivo HTML de 442 bytes		Archivo PHP de 218 bytes	
	Promedio de tiempo (segundos)	Memoria utilizada en bytes	Promedio de tiempo (segundos)	Memoria utilizada en bytes	Promedio de tiempo (segundos)	Memoria utilizada en bytes
Include	2.70	616	1.71	616	0.75	848
require	2.30	616	1.78	616	0.72	848
include_once	0.40	496	0.26	496	0.0186	848
require_once	0.38	496	0.25	496	0.0188	848

Aunque los resultados nos indiquen qué sentencia es la más eficiente en responder o la que ocupa menos recursos, para una ejecución exitosa de nuestro programa se tiene que tomar en cuenta qué tipo de archivo se va a llamar y si este afectará de algún modo nuestro código, el número de veces que se llamará y empatarlo con el comportamiento de la sentencia que convenga, en caso de haber al menos dos posibilidades entonces se puede tomar en cuenta la eficiencia del tiempo y memoria que requiere.

Codificación de caracteres

Emplear UTF-8 sin BOM (siglas en inglés de *byte order mark*, es un caracter que indica el orden de los bytes y la codificación que se usa: UTF-8, UTF-16 o UTF-32) en lugar de ANSI u otra codificación es mucho mejor, debido a que un determinado carácter (como una eñe, por ejemplo) en otra codificación puede ser algo completamente distinto. Si los usuarios emplean la misma codificación no habría problema pero, si en un momento dado se entra a la página en una computadora de algún usuario que está bajo otra configuración, por ejemplo en ruso, verá la pantalla llena de caracteres sin sentido.

Así, es más conveniente el uso de esta codificación para cubrir el caso en el que la página estuviera disponible en más de un idioma, incluyendo la codificación y colación de la base de datos.

Uso de variables globales.

En PHP, el contexto de las variables usadas en un programa es de forma local, esto también influye al incluir código desde otro archivo. No obstante, si en el interior de una función se requiere llamar a una variable que haya sido definida en otra sección del mismo programa, se puede utilizar la palabra reservada *global* o hacer uso de la variable *$GLOBALS*; la primera es anteponiendo la palabra ante las variables (definiendo una por una o separando las variables mediante una coma) a llamar y la segunda, es una las variables (predefinida en PHP) que se consideran como "superglobales", es decir, *$GLOBALS* es un arreglo asociativo que contiene todas las referencias a todas las variables que están disponibles en el contexto global dentro del programa donde los nombres de las variables son las claves del arreglo. Se puede observar a modo de ejemplo el **Código 20** del **Anexo**.

La práctica del uso de variables globales debe estar limitada, ya que tanto la palabra reservada *global* como la variable *$GLOBALS* causa un costo de tiempo y memoria mayor que al hacerlo con variables locales. Por ejemplo, mediante el uso del **Código 16**, **Código 17** y **Código 18** del **Anexo**, obtengo los siguientes resultados al ejecutar cualquiera de las funciones previamente definidas del **Código 21** del **Anexo** para comparar la eficiencia de su ejecución:

Función	**Memoria usada**	**Promedio de tiempo en segundos utilizado al llamar la función:**			
		3 000 veces	**30 000 veces**	**300 000 veces**	**600 000 veces**
sinGlobalArg()	168 bytes	0.002886902	0.029425653	0.287310129	0.594494922
sinGlobal()	168 bytes	0.003516396	0.034755491	0.350045282	0.787137017
conGlobal()	168 bytes	0.003912888	0.038725175	0.396971842	0.905323585
conGlobals()	176 bytes	0.004124545	0.042008784	0.427999517	0.976938198

El uso de memoria es mayor cuando se utiliza la variable *$GLOBALS* de PHP en comparación a las otras funciones, la función *sinGlobalArg()* es la más rápida en responder al incluir la variable como argumento que las otras funciones a menos que sea necesario tener variables globales el uso de estos recursos debe ser limitado.

Configuración en un archivo.

A medida que la aplicación va creciendo, siempre será necesario acceder en distintos lugares a determinados valores. Entonces, el desarrollador debe tomar en cuenta que debe almacenarlos todos en un único lugar, así evitará tener que modificar todos los archivos cada vez que haya un cambio. Un ejemplo, es como si la aplicación fuera un carrito de compras y que en un determinado momento hay que cambiar el IVA.

Comprobación de variables.

La construcción del lenguaje *isset()*, la cual determina si una variable está definida y no es la constante *NULL*, resulta de una gran utilidad. Sirve tanto para saber si una determinada variable ha sido inicializada como para comprobar que un índice de un arreglo existe e incluso para trabajar con longitudes de cadenas de un modo más eficiente: si es que en algún determinado momento se necesita comprobar que una cadena tiene al menos una determinada longitud. Por ejemplo que el nombre de usuario tenga más de cuatro caracteres, lo más inmediato que se me podría ocurrir es utilizar *strlen()* para obtener la longitud de la cadena de la siguiente manera:

```
<?php

    if ( strlen($username) < 5 ){
          // Bloque de código.
    }

?>
```

Pero, en tanto que *strlen()* es una función, PHP necesita realizar un trabajo previo para ejecutar la llamada (convertirla a minúsculas para luego buscarla en la tabla hash de funciones), además de ejecutar la propia función. Para fines más prácticos, es mejor usar el constructor del lenguaje *isset*. Esto significa que PHP no tendrá que hacer ninguna operación previa ni habrá sobrecarga, por lo que si se quiere comprobar que una variable tiene más de una determinada longitud, se puede hacer más efectivo y eficiente sin gastar recursos innecesarios:

```
<?php

    if ( isset($username{5}) ){
        // Bloque de código.
    }

?>
```

Separar y reutilizar código.

Se debe evitar tener archivos con cientos o miles de líneas, lo más recomendable es agrupar las funciones que se vayan a emplear con frecuencia en ciertos archivos e incluirlos para futura reutilización. A su vez, recordar el concepto DRY (*Don't repeat yourself*) que se aplica para todos los lenguajes de programación, se debe parametrizar todo lo posible y así evitar errores absurdos.

Documentación.

Me parece muy recomendable hacer uso de los comentarios en el código, de preferencia se debe usar un formato estándar como el de JavaDoc o PHPDoc, aunque existen diversas aplicaciones que pueden hacer uso del mismo. Se debe comentar partes del código que resulten relevantes, evitar detalles obvios como poner antes del ciclo: "Recorro el arreglo…", los comentarios suelen ser de gran ayuda pero podrían llegar a estorbar si no se usan bien. Cuando no se realizan este tipo de prácticas, es muy común que después de seis meses se olvide el código que uno genera (particularmente si no se tiene un estándar para la generación del mismo), al igual que suele crear conflictos con otros desarrolladores que requieran utilizarlo.

Dar estructura coherente al código.

Se debe tener la costumbre de estructurar el código en distintos componentes, eso ayudará a realizar cambios en un futuro próximo. En la arquitectura de software, el patrón más común o similar en PHP es el MVC (Modelo-Vista-Controlador) donde se recomienda que el código se debe estructurar en tres grandes grupos. Por lo general en cualquier aplicación desarrollada sobre PHP se realiza una conexión con una base de datos, se implementa una lógica de negocio y por último una implementación de HTML que se va a mostrar en el navegador. Esta debe ser la forma de trabajar para no mezclar el código. Como ejemplo práctico, si se quiere mostrar una tabla con los resultados de una consulta se tendría que estructurar de la siguiente manera:

- Primero se realizan la parametrización de la base de datos y la conexión con el servidor (Controlador).
- En la lógica de negocio inicializaremos un arreglo con los resultados y los valores que se quieren mostrar (Modelo).
- Se recorre el arreglo creando una tabla para mostrarla al usuario (Vista).

CONCLUSIONES

Gracias a los resultados obtenidos por el primer sistema de evaluación académica (*EVA 2009-2*), dio inicio a la repetición del mismo por cada periodo semestral siguiente para mantener esta necesitad. Por medio de este avance, se dio inicio a considerar otros procesos que requerían un mejor manejo de información para cambiar parte de los trámites que anteriormente se solicitaban en papel, tales como: solicitud de asignaturas inscritas, información del alumno, listado de grupos, entre otros. Conforme los requerimientos tomaban cierto grado de importancia, comenzaron a realizar estos cambios para mitigar la demanda en esos momentos, aunque esto no fue lo adecuado debido a que al darle mantenimiento o realizar una modificación de forma exitosa implicaba una inversión de tiempo mayor en el periodo cursado con respecto al anterior por el o los nuevos sistemas que hayan sido replicados, si uno requería ser modificado entonces para mantener su similitud con todos los demás que hayan sido copiados, deberían de modificarse ya que de lo contrario podría acarrear problemas inesperados.

Adicionalmente, otro conflicto es la forma de almacenar la información. Esto es que la carencia del desarrollo de algunas tablas en su base de datos, la inconsistencia de su información era un foco de atención para resultados que podrían ser graves a la hora de procesar o solicitar parte de la información.

Con el crecimiento de la información y de algunos sistemas, dio comienzo a que realizara un análisis para dar una solución al encontrar la similitud de los sistemas y darle soporte a uno solo que realizar lo mismo a varios. encontrar una solución para evitar que siguieran multiplicándose los que uno sólo puede realizar dicha labor conflictos que pudieran ocurrir, comenzando por algunas bases de datos para reducirlas en cuanto a datos o registros repetidos para una respuesta más eficiente a la hora de ser solicitados.

Gracias a los diversos cambios al sistema de evaluación académica (EVA) en la Facultad de Música, se ha reducido el espacio utilizado en el servidor Web al evitar siete sistemas con el mismo comportamiento y requerir de uno solo, para mejorar el tiempo de respuesta al no precargar funciones repetidas y únicamente contar con las necesarias, de la misma manera al implementar los detalles de las buenas prácticas para mejorar la velocidad de respuesta a las solicitudes. Las mejoras que realicé, han hecho que el código sea más ligero y legible con la documentación de la que se carecía. Dentro de la dinámica laboral, he unificado o modificado aplicaciones que deban ser actualizadas y que las nuevas aplicaciones se apeguen a la estructura que actualmente se está utilizando.

Algunos de los conocimientos adquiridos en la carrera que me ayudaron a desempeñarme en mis funciones laborales fueron:

- → Desde el análisis y aprendizaje de las nuevas tecnologías como en este caso fue el lenguaje de programación PHP (visto en Matemáticas Discretas, Introducción a las Ciencias de la Computación I, Análisis Lógico, Arquitectura de Computadoras e Inteligencia Artificial) y que me ayuda a mantener actualizado para cualquier otro lenguaje que se presente.
- → El conocimiento de diversos problemas que tienen soluciones óptimas (visto en Análisis de Algoritmos) para evitar caer en la reinvención de la rueda o generar una solución menos eficiente

- → El buen manejo de las estructuras de datos (visto en Introducción a las Ciencias de la Computación II) para mantener una complejidad baja en la creación de mis propias funciones.
- → Conocimiento y dominio de los principales conceptos al diseño, construcción y uso eficiente de las bases de datos (visto en Sistemas de Bases de Datos).
- → El uso de mi análisis y el gusto de la realización del buen desarrollo de la programación para una mejor formalidad y entendimiento que desarrollé durante toda la carrera.

Para de la información se volvió más consistente y se ha previsto parte de los conflictos principales como el envío de información con la DGAE. Se tiene como objetivo terminar de completar la consistencia de la información restante, verificar si se necesitan crear nuevas tablas, modificar las tablas existentes o eliminar tablas que dejaron de usarse.

De la misma forma, mediante el proceso para el desarrollo del software se procurará unificar todos los sistemas existentes en uno sólo de tal manera que al seguir el patrón MVC (Modelo-Vista-Controlador), realizar el mismo análisis para concentrar todas las funciones que fueron creadas y unificarlas a modo que sólo habrá un único modelo que mantendrá la gestión de todo el acceso de dicha información, crear un sistema más interactiva de acuerdo al tipo de usuario que tenga acceso. Por último, este sistema será flexible para poder ingresar nuevos procesos a los nuevos requerimientos que surjan conforme a la demanda que la Facultad de Música lo requiera.

BIBLIOGRAFÍA

Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). Introduction to algorithms (third edition). The MIT Press Cambridge, Massachusetts.

Kendall y Kendall (2011) Análisis y diseño de sistemas, Pearson Educación, México

Pressman, Roger S. (2010). Ingeniería de software. Un enfoque práctico (Séptima edición). McGraw-Hill

http://php.net/manual/es/

http://dev.mysql.com/doc/refman/5.0/es/

Anexo

Información detallada de los programas que se usaron como ejemplos y soluciones para una mejor eficiencia.

```
function conexion($BD){

    // Realizo la conexión
    $conexion = mysql_connect("localhost","root","");

    // Si la conexión no es satisfactoria, mando mensaje de error
    if (!$conexion){
        die('No se pudo conectar: '.mysql_error());
    }

    // Selecciono la base de datos mediante la conexión
    mysql_select_db($BD, $conexion);

    // Devuelvo el resultado de la conexión
    return($conexion);

}

// Realizo la conexión
$conexion = conexion('nombre_de_la_BD');


/*
 * Realizo la consulta pidiendo RFC y nombre de los profesores
 * pidiendo que en el RFC contenga al menos un dígito mediante
 * expresiones regulares y pido que este listado se ordene
 * alfabéticamente mediante el nombre.
 */

$consultaProfesores = mysql_query("SELECT rfc, nombre FROM
tabla_profesores WHERE rfc REGEXP '[0-9]' ORDER BY nombre",
$conexion) or die (mysql_error());
```

Código 1. Conexión y consulta de profesores.

```
// Mientras haya registros en la consulta
while ($rowProf = mysql_fetch_array($consultaProfesores)) {
    $rfcFam = $rowProf['rfc'];
    // CODIFICO DE ISO-8859-1 A UTF-8 Y QUITO ACENTOS
    $nombreFam = normaliza( utf8_encode($rowProf['nombre']) );

    // Realizamos la búsqueda a la lista de la DGAE
    $clave = array_search($nombreFam, $profesDgae);

    // Veo si hay más de una clave.
    $homonimos = array_keys($profesDgae, $nombreFam);

    // Si la clave es diferente de FALSE y el tamaño del
    // arreglo $homonimos es uno, entonces ese RFC se almacena
    if( $clave !== FALSE && count($homonimos) == 1 ){
        $transformacionFamDgae[]    =    Array($rfcFam,    $nombreFam,
$clave, $profesDgae[$clave]);
        // Elimino la clave encontrada
        unset($profesDgae[$clave]);
    } else{
       // En otro caso, lo guardo para el otro comparativo
        $profesFam[$rfcFam] = $nombreFam;
    }
}
```

Código 2. Primer comparativo de profesores.

```
     // si no hay errores registramos al usuario
     if ( empty($error) ) {

          $query1  = "UPDATE TABLA SET MODIFICA LA CALIFICACION ";
          $query1 .= "WHERE CONDICION PARA LA MODIFICACION";

          ejecutaConsulta($query1);
                       $brinca = "Location: REDIRECCIONA";

          header( $brinca );
          die;

     }
```

Código 3. Actualiza calificaciones.

```
function muestraError($resultadoConsulta, $activaMensaje = FALSE){
    /*
     * Si $activaMensaje es TRUE, mostramos el origen del error
     * en la consulta; si no, mostramos que hubo un problema
     * general y detenemos el flujo de la operación.
     */
    if ( !$resultadoConsulta && $activaMensaje ) {
        die ('Consulta no v&aacute;lida: <p />' . mysql_error());
    } elseif ( !$resultadoConsulta ){
        die ('Ha ocurrido un problema con tu solicitud,
int&eacute;ntelom&aacute;s tarde.');
    }
}
```

Código 4. Función que indica si ocurrió algún error.

```
function consultaAArreglo($resultado, $tipoArreglo){
    $tipoArreglo = strtolower($tipoArreglo);
    // Vemos qué tipo arreglo se necesita.
    switch($tipoArreglo){
        Case 'n':
            $parametro = MYSQL_NUM;
        break;
        Case 'a':
            $parametro = MYSQL_ASSOC;
        break;
        Case 'b':
        default :
            $parametro = MYSQL_BOTH;
    }

    $arreglo = Array();
    while ($fila = mysql_fetch_array($resultado, $parametro)) {
        $arreglo[] = $fila;
    }
    return $arreglo;
}
```

Código 5. Función que convierte una consulta MySQL a un arreglo.

```
function consulta($tipo, $tabla, $columnas = '*', $condicion = '1',
$tipoArreglo = 'a') {

    $consulta = '';

    $tipo = strtolower($tipo);

    // Construyo la cadena de consulta de acuerdo al tipo.
    switch($tipo){

        Case 'a':
            $consulta = "UPDATE $tabla SET $columnas WHERE $condicion;";
        break;

        Case 'i':
            $claves = implode(', ', array_keys($columnas));
            $valores = '\''.implode('\', \'', $columnas).'\'';
            $consulta = "INSERT INTO $tabla($claves) VALUES ($valores);";
        break;

        Case 'b':
            $consulta = "DELETE FROM $tabla WHERE $columnas;";
        break;

        Case 's':
            $consulta = "SELECT $columnas FROM $tabla WHERE $condicion;";
        break;

    }

    // Realiza la conexión con MySQL.
    $conexion = conectar();

    $resultado = mysql_query ($consulta, $conexion);

    // Verifico si ocurrió algún error.
    muestraError($resultado, TRUE);

    // Transformo la consulta en un arreglo si es el caso SELECT.
    $datos = ( $tipo == 's' ) ? consultaAArreglo($resultado,
$tipoArreglo) : TRUE;

    // Cerramos la conexión y devolvemos el arreglo.
    mysql_close($conexion);

    return $datos;

}
```

Código 6. Función para ejecutar una consulta de tipo UPDATE, INSERT, DELECT o SELECT.

```php
<?php
    function do_html_URL($urlU, $nameU){
        // output URL as link and br
?>
        <a href = "<?php echo $urlU; ?>"><?php echo $nameU; ?></a>
<?php
    }
?>
```

Código 7. Función do_html_URL().

```
function NOMBRE DE LA FUNCIÓN($reg_array){
    //display all books in the array passed in
if (!is_array($reg_array)){
echo '<br />No hay registros<br />';
     return;
  } else  {
    //create table
echo '<table width = \"100%\" border = 1>';
echo '<tr><td>';
     echo '<font color="#666666">TÍTULO DE LA COLUMNA 1</font>';
echo '</td><td>';
     echo '<font color="#666666">TÍTULO DE LACOLUMNA 2</font>';
echo '</td><td>';
                                    ...
     echo '<font color="#666666">TÍTULO DE LACOLUMNA iÉSIMA</font>';
echo '</td><td>';
                                    ...
     echo '<font color="#666666">TÍTULO DE LACOLUMNA N</font>';
echo '</td></tr>';
foreach ($reg_array as $row){
echo '<tr><td>';
     echo  $row[CLAVE DE LA COLUMNA 1];
echo '</td><td>';
     echo $row[CLAVE DE LA COLUMNA 2];
echo '</td><td>';
                                    ...
     $url = HIPERVÍNCULO CON O SIN DATOS A ENVIAR POR EL MÉTODO GET;
           $title = $row[CLAVE DE LA COLUMNA iÉSIMA];
     do_html_url($url, $title);
echo '</td><td>';
                                    ...
     echo $row[CLAVE DE LA COLUMNA N];
echo '</td></tr>';
}
echo '</table>';
  }
echo '<hr />';
}
```

Código 8. Función para crear una tabla con hipervínculos.

```
function NOMBRE DE LA FUNCIÓN($reg_array){
     //display all books in the array passed in
     if (!is_array($reg_array))
     {
     echo '<br />No hay registros<br />';
          return;
       }
     else
       {
         //create table
     echo '<table width = \"100%\" border = 1>';
     echo '<tr><td>';
          echo '<font color="#666666">COLUMNA 1</font>';
     echo '</td><td>';
          echo '<font color="#666666">COLUMNA 2</font>';
     echo '</td><tr>';
                                        ...
          echo '<font color="#666666">COLUMNA N</font>';
     echo '</td></tr>';
     foreach ($reg_array as $row){
     echo '<tr><td>';
          echo $row[CLAVE DE LA COLUMNA 1];
     echo '</td><td>';
          echo $row[CLAVE DE LA COLUMNA 2];
     echo '</td><td>';
                                        ...
          echo$row[CLAVE DE LA COLUMNA N];
     echo '</td></tr>';
     }
     echo '</table>';
       }
     echo '<hr />';
}
```

Código 9. Función para crear una tabla sin hipervínculos.

```
<?php
     $arreglo = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];

     // Si se une por una coma (',').
     echo implode(',', $arreglo); // Muestra: 0,1,2,3,4,5,6,7,8,9

     // Si se une por la cadena vacía ('').
     echo implode('', $arreglo); // Muestra: 0123456789

     // Si sólo se pasa el arreglo, es equivalente
     // a la ejecución anterior por cadena vacía ('').
     echo implode($arreglo); // Muestra: 0123456789
?>
```

Código 10. Ejemplo de la función implode().

```
function muestraTabla($registros, $encabezado = Array()){
    $tabla = '<table border = "1">';
    if( $encabezado !== Array() ){
        $tabla .= '<thead><th>' . implode('</th><th>', $encabezado)
. '</th></thead>';
    }
    $tabla .= '<tbody>';
    if (!is_array($registros)){
        $tabla .=  '<tr><td><br />No hay registros<br /></td></tr>';
    } else{
        foreach ($registros as $datosPorRegistro) {
            $tabla    .=    '<tr><td>'    .    implode('</td><td>',
$datosPorRegistro) . '</td></tr>';
        }
    }
    $tabla .= '</tbody></table>';
    echo $tabla;
}
```

Código 11. Función para mostrar una tabla en HTML.

```
          function do_html_URL($urlU, $nameU){
               return "<a href = \"$urlU\">$nameU</a><br />";
          }
```

Código 12. Modificación de la función do_html_URL().

```
function hipervinculos($arreglo, $columna, $enlace, $variablesPorGet
= Array()){

    foreach($arreglo as $iArreglo => $registros){
        $cadenaDeVariables = '';
        if($variablesPorGet!== Array() ){
            $variables = Array();
            foreach($variablesPorGet as $indice => $valor){
                $variables[] = $indice.'='.$valor;
            }

            if( $variables !== Array() ){
                $cadenaDeVariables = '?' . implode('&', $variables);
            }
        }
        $url = $enlace.$cadenaDeVariables;
        $title = $arreglo[$iArreglo][$columna];
        $arreglo[$iArreglo][$columna] = do_html_url($url, $title);
    }
    return $arreglo;

}
```

Código 13. Función hipervinculos().

```
function getSemestre($nSemestre = 0, $anio = NULL){

    date_default_timezone_set('America/Mexico_City');

    $anio = ( isset($anio) ) ? $anio : (int)date('Y');
    $mes = (int)date('n');
    $dia = (int)date('d');

    $digito = 1;

    ( ($mes == 6 && $dia> 15) || $mes > 6 ) ? ++$anio : ++$digito;

    $subDigito = (int)($nSemestre % 2);

    $anio += (int)($nSemestre / 2);

    if($digito === 1 && $subDigito< 0){

        --$anio;
        ++$digito;

    } elseif($digito === 2 && 0 < $subDigito){

        ++$anio;
        --$digito;

    } else{

        $digito += $subDigito;

    }

    return (string)"$anio-$digito";

}
```

Código 14. Función getSemestre().

```
                function arregloPeriodos($total){

                    $periodos = Array();
                    for($n = $total; $n >= 0; --$n){

                        $periodos[] = getSemestre(-$n);

                    }
                    return $periodos;

                }
```

Código 15. Función arregloPeriodos().

```php
<?php

    function microtime_float(){

        list($useg, $seg) = explode(" ", microtime());

        return ((float)$useg + (float)$seg);

    }

?>
```

Código 16. Función microtime_float().

```php
<?php

    // Indico el número de veces que haga la medición.
    $repeticiones = 10;

    // Indico el número de ejecuciones del código a medir.
    $llamadas = 1000;

    for ($i = 0; $i < $repeticiones; ++$i){

        // Calculo el tiempo inicial.
        $tiempoInicial = microtime_float();

        for ($n = 0; $n < $llamadas; ++$n){

            // Bloque de código a llamar.

        }

        // Calculo el tiempo final.
        $tiempoFinal = microtime_float();

        // Calculo la diferencia del tiempo final con el inicial.
        $diferenciaDeTiempo = $tiempoFinal  - $tiempoInicial;

        // Muestro el resultado de la medición de tiempo.
        echo 'Tiempo empleado: ' . $diferenciaDeTiempo;
        echo ' segundos.<br />';

    }

?>
```

Código 17. Estructura para medir el tiempo de un programa.

```php
<?php

    // Indico el número de veces que haga la medición.
    $repeticiones = 10;

    // Indico el número de ejecuciones del código a medir.
    $llamadas = 1000;

    for ($i = 0; $i < $repeticiones; ++$i){

        // Calculo la memoria inicial.
        $memoriaInicial = memory_get_usage();

        for ($n = 0; $n < $llamadas; ++$n){

            // Bloque de código a llamar.

        }

        // Calculo la memoria final.
        $memoriaFinal = memory_get_usage();

        // Calculo la diferencia de la memoria final con la inicial.
        $diferenciaDeMemoria = $memoriaFinal - $memoriaInicial;

        // Muestro el resultado de la medición de memoria.
        echo 'Memoria empleada: ' . $diferenciaDeMemoria;
        echo ' bytes.<br />';
    }

?>
```

Código 18. Estructura para medir la memoria de un programa.

```php
<?php

    echo '<br />Archivo llamado ' . $archivo . '<br />';

    for ($i = 0; $i < 10000; ++$i){

        switch ($sentencia){

            case 1:
                include $archivo;
                break;

            case 2:
                require $archivo;
                break;

            case 3:
                include_once $archivo;
                break;

            case 4:
                require_once $archivo;
                break;

        }

    }

    echo 'Sentencia <strong>'.$s.'</strong><br />';

?>
```

Código 19. Importación de un archivo.

```php
<?php

    // Variable global de todo el pograma
    $variableGlobal = "¡Hola!";

    function f1(){
        echo $variableGlobal;
    }

    function f2(){
        // Hacemos una referencia global a $variableGlobal.
        global $variableGlobal;
        echo $variableGlobal;
    }

    function f3(){
        // Imprime el valor de referencia global a la clave
        // asignada como variableGlobal del arreglo $GLOBALS
        echo $GLOBALS['variableGlobal'];
    }

    // Como la variable no está definida, no hace nada.
    f1();

    // Imprime el mensaje: ¡Hola!
    f2();

    // Imprime el mensaje: ¡Hola!
    f3();

?>
```

Código 20. Uso de variables globales.

```php
<?php

    $a = 2;
    function sinGlobalArg($a) {
        //$a = 2;
        $b = 'casa';
               $a;
        ++$a;
        $c = $a . $b;
        $d = 48;
        $d += 72;
        $d = $c . $d . $b;
    }

    function sinGlobal() {
        $a = 2;
        $b = 'casa';
        $a;
        ++$a;
        $c = $a . $b;
        $d = 48;
        $d += 72;
        $d = $c . $d . $b;
    }

    function conGlobal() {
        $a = 2;
        $b = 'casa';
        global $a;
        ++$a;
        $c = $a . $b;
        $d = 48;
        $d += 72;
        $d = $c . $d . $b;
    }

    function conGlobals() {
        $a = 2;
        $b = 'casa';
        ++$GLOBALS['a'];
        $c = $GLOBALS['a'] . $b;
        $d = 48;
        $d += 72;
        $d = $c . $d . $b;
    }

?>
```

Código 21. Funciones con uso global.