

MANUAL DEL ALUMNO

INSTRUCTOR: ING. CARLOS ZARAGOZA ORTIZ

►Taller de aplicaciones RIA

Enfoque práctico del desarrollo de un proyecto web

Ing. Carlos Zaragoza Ortiz ► ing.zaragoza@yahoo.com

Temario

MODULO 1 MYSQL 5.0

- ▶ TALLER DE APLICACIONES RIA INTRODUCCIÓN
- ▶ ARTES GRAFICAS DE TIJUANA A.C.
 - ▶ PLANTEAMIENTO
 - ▶ NECESIDADES A SOLUCIONAR
- ▶ TODO EN UN SITIO
 - ▶ IMPLEMENTANDO SERVIDOR
 - ▶ INSTALANDO EL ADMINISTRADOR DE MYSQL
- ▶ CREANDO BASE DE DATOS
- ▶ GENERANDO PROCEDIMIENTOS ALMACENADOS
 - ▶ TABLAS DE PERMISOS
 - ▶ DECLARACION DE VARIABLES LOCALES
 - ▶ SENTENCIA SET
 - ▶ SENTENCIA IF
 - ▶ PROCEDIMIENTO PARA GUARDAR USUARIO DEL SISTEMA
 - ▶ GENERANDO DEMAS PROCEDIMIENTOS
- ▶ TRIGGERS
 - ▶ SINTAXIS
 - ▶ CREANDO TRIGGERS PARA NUESTRAS TABLAS
- ▶ VISTAS EN MYSQL
 - ▶ SINTAXIS
 - ▶ CREANDO VISTAS PARA EL SISTEMA

MODULO 2 FLEX 3

- ▶ DISEÑO DE LA INTERFAZ CON ADOBE FLEX 3
 - ▶ CREANDO Y CONFIGURANDO NUESTRO PROYECTO FLEX
 - ▶ CONFIGURACIÓN DE CARPETA BIN-DEBUG
 - ▶ CREAR NUESTRO LOGGIN

- ▶ CREAR MENU PRINCIPAL PARA ADMINISTRADORES
- ▶ CREAR COMPONENTES
 - ▶ ALTA DE ALUMNOS
 - ▶ SUBIR ARCHIVO (FOTOS)
 - ▶ INSCRIBIR ALUMNOS AL SISTEMA
 - ▶ USO DEL DATAGRID
 - ▶ CATALOGOS PARA EL SISTEMA
 - ▶ FILTRADO DE ALUMNOS INSCRITOS
 - ▶ CREANDO RESULTADOS DE PRIMER ESTADO

MODULO 3 REPORTES Y GRAFICAS

- ▶ INSTALACIÓN DE ENTORNO DE DESARROLLO
- ▶ CONOCIENDO IREPORTS
 - ▶ CREACIÓN DE REPORTE ENLAZADO A BASE DE DATOS
- ▶ CREACIÓN DE SERVLET PARA RESOLVER LLAMADAS A NUESTRO REPORTE
- ▶ AGREGANDO PARAMETROS A NUESTRO REPORTE
- ▶ LLAMANDO PARAMETROS DESDE EL SERVLET
- ▶ GRAFICAS EN FLEX
 - ▶ CREANDO UN CHART
 - ▶ GENERANDO XML
 - ▶ IMPORTANDO DATOS AL CHART
 - ▶ FORMATEANDO EJES
 - ▶ USANDO DATATIPS
 - ▶ IMPLEMENTANDO EFECTOS
 - ▶ UTILIZANDO DOS SERIES EN UNA GRAFICA.

Taller de aplicaciones RIA

Enfoque práctico del desarrollo de un proyecto web

Actualmente las aplicaciones web han crecido en una manera que no cada día surgen más herramientas que nos ayudan a tener proyectos robustos y con mejor rendimiento.

En la actualidad Java Script ha sido un lenguaje del lado del cliente que nos ha permitido tener todo un conjunto de elementos que nos permiten desde crear una conexión a una base de datos haciendo uso de una tecnología del lado del servidor, hasta crear interacciones entre el usuario y la página con eventos DOM.

Vemos cada día como estas herramientas están siendo utilizadas cada vez más y vemos nuevas herramientas que agilizan estas tareas, haciendo la experiencia de desarrollo mucho más rápida y efectiva.

Hablamos de la nueva plataforma Adobe Flex 3 que con un sinfín de componentes que podemos utilizar en nuestras aplicaciones, obtenemos un resultado profesional y veloz, no demeritando la calidad.

Pero Flex 3 no es todo el conjunto, ya que este solo es el programa que nos permite tener un entorno del lado del cliente mucho mejor, trasladando la experiencia del usuario desktop a las comunicaciones web, y por medio de herramientas del lado del servidor como lo es MYSQL y PHP, logramos tener interactividad y dinamismo, simpleza y potencia, rentabilidad y escalabilidad.

Conclusión

Hoy en día las tecnologías nos han permitido crear entornos mucho más agradables para los clientes finales y hacer que estas aplicaciones sean de desarrollo rápido y rico es lo que ha dotado al internet actual de tantas características donde el centro es el usuario o visitante de internet.

ARTES GRAFICAS DE TIJUANA A.C.

Descripción de las necesidades de la aplicación

Planteamiento

El desarrollo de la tecnología ha ido evolucionando en el contexto mundial, dando alcance por medio de la diversificación de medios de comunicación masivos al acceso a información que hace 10 años no se tenía, esto ha ido evolucionando y la necesidad por estar a la vanguardia tecnológica no se deja de tener en cuenta como prioridad dentro de las instituciones de cualquier índole.

El instituto de Artes Graficas A.C. de la ciudad de Tijuana busca una manera de llevar su información de alumnos y asignación de materias con profesor a un sistema en un entorno web, la necesidad ha crecido, ya que la institución tiene en operación desde hace ya cuatro años, sin embargo, la manera en la que ha realizado inscripciones, altas, bajas, colegiaturas y registro de personal ha sido de manera manual, esto ha significado procesos decrecientes al ritmo de la institución.

La institución cuenta actualmente con una matrícula de 260 alumnos de los cuales están divididos en diferentes programas o cursos, estos son los siguientes:

- ▶ Pintura
- ▶ Música
- ▶ Danza
- ▶ Escultura

La plantilla de personal es de 12 profesores de asignatura, cuatro directores de carrera, subdirector y el director, así como personal administrativo en los cuales

está el jefe de finanzas, departamento jurídico y control escolar; en este último es donde se ha aplicado el análisis para saber que necesidades presenta la institución para solución.

Necesidades a solucionar

Control de los alumnos: La institución está abierta al público en general con la condicional de que sean personas mayores a 15 años. Cada una de ellas debe cubrir los siguientes requisitos para poder inscribirse en la institución.

- ▶ Comprobante de domicilio
- ▶ Acta de nacimiento
- ▶ Constancia de estudios cursados o en curso
- ▶ Fotografías (6 B/N)
- ▶ Aportar la cantidad de \$1500.00 a la cuenta bancaria 324423-643 Banco de México a nombre de: Artes Graficas de Tijuana A.C.

Una vez que cubren con esos requisitos, debe ser llevados a la ventanilla de control escolar para ser recibidos y archivados, si cumple con todos los requisitos, se guardar en una hoja de Excel de alumnos inscritos, ya dentro de esta hoja, se solicita el nombre del curso o programa a cursar y se le dice al alumno que regrese en fecha próxima para revisar sus horarios y materias a cursar.

El proceso que sigue es el de adecuar a los nuevos alumnos a los cursos que estarán inscritos para ver que profesores darán las materias y en que horarios.

Si el alumno ya está inscrito, solo se requiere su matrícula para hacer una reinscripción al semestre que le corresponde dependiendo del curso.

Control de los profesores: Ellos se contratan dependiendo del perfil requerido y una vez que pasan la entrevista, se les pide la siguiente documentación:

- ▶ Solicitud de empleo
- ▶ Comprobante de domicilio
- ▶ Historial medico
- ▶ C.V.

- ▶ RFC
- ▶ Hoja de inscripción a la institución

Una vez entregados los archivan y registran en una hoja de Excel el nombre del profesor y la materia o materias que va a dar.

Control de asignación de materias a alumnos: Este proceso sucede cuando ya se tiene el personal completo y listo en las listas de materia-profesor ya que ahí se asigna el grupo que estará tomando el curso. Una vez que se sabe esto se asigna la matrícula y el nombre del curso con la condicional que de que no sean más de 25 alumnos por grupo. Estos grupos son asignados dependiendo el periodo y el año en que se van a impartir.

Las cuentas de los alumnos por conceptos de:

- ▶ Inscripción
- ▶ Reinscripción
- ▶ Materias reprobadas
- ▶ Material didáctico
- ▶ Libros
- ▶ Visitas guiadas
- ▶ Conciertos
- ▶ Eventos culturales

Son llevadas en el control de finanzas ya que ellos son los que cobrar cada uno de estos conceptos. Cuando se da de alta un alumno esa cuenta se crea en finanzas para poder llevar el control de todo lo que ha pagado y debe.

Los reportes que libera la institución son:

- ▶ Acta de calificaciones parcial
 - ▶ Boleta semestral
 - ▶ Historial académico
 - ▶ Recibo por pago de colegiatura y diversos conceptos
 - ▶ Lista de profesores y materias
-

- ▶ Lista de alumnos asignados al grupo
- ▶ Listas para profesores
- ▶ Acta de calificaciones finales

Este es el proceso que se lleva actualmente en la institución, lo que ahora nos corresponde es poder llevar esto a una base de datos, después a una plataforma web y poder administrar esta información.

TODO EN UN SOLO SITIO

Implementando server

El primer proceso será implementar un servidor que nos proporcione la capacidad de publicar páginas PHP, para esto podemos utilizar el proyecto WAMP si es que estamos trabajando para Windows, sin embargo esto no es limitante, ya que este servidor es de desarrollo, cuando se requiera uno para producción las condiciones de hardware y software son diferentes.

El primer paso es conseguir el programa e instalarlo de la dirección:

<http://www.wampserver.com/en/>

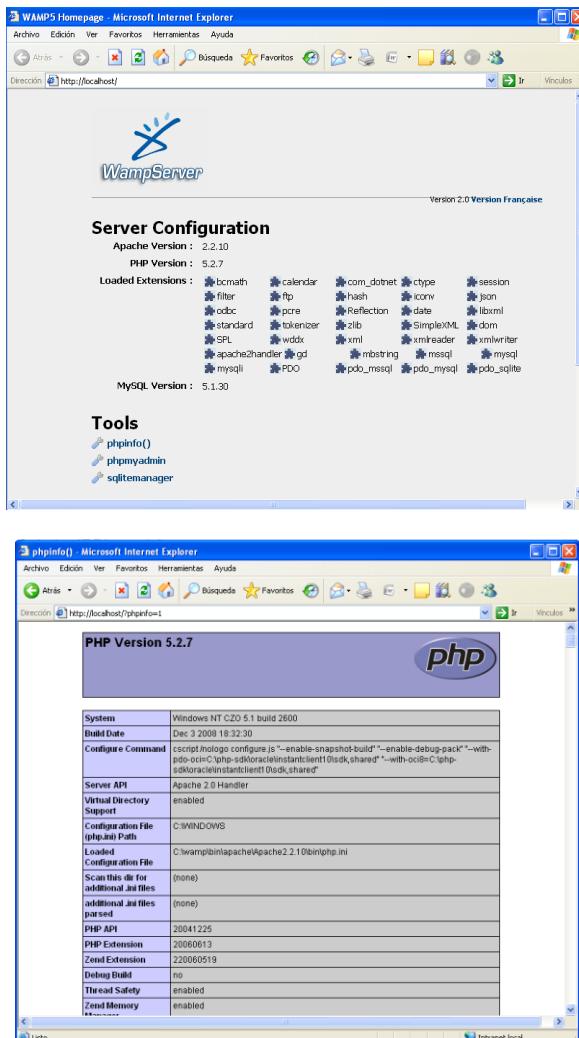
ya que tenemos el software en nuestro equipo hay que ejecutarlo para instalación la cual nos presentará este primer cuadro de dialogo:



Una vez que abra, hay que seguir el asistente que nos ayudara para configurar nuestro servidor SMTP, por ahora nosotros dejaremos esta sección como esta es decir **localhost** como servidor.



Esta imagen podemos verla del lado derecho junto al reloj esta un símbolo de forma de media luna, ese es quien nos permite tenerlo y poder usarlo, para probar que todo está bien, solo hay que ir a localhost y después a phpinfo. Una vez funcionando tendremos el correcto desempeño de nuestras aplicaciones. Esta herramienta ya trae consigo Mysql + PHP + Apache, es decir que tenemos un pequeño sistema listo para iniciar con nuestra base de datos.



Administrador de MySQL

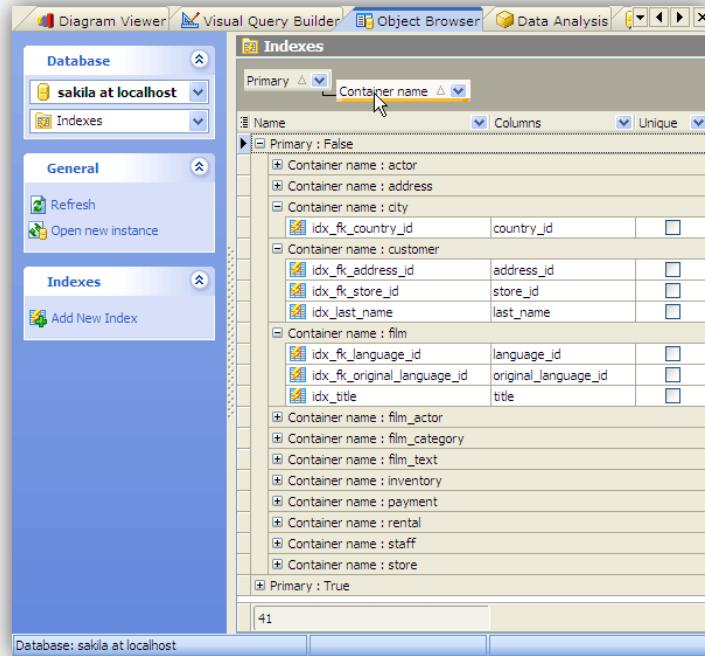
Es esta parte de la iniciación del proyecto necesitamos descargar una herramienta como MySQL GUI que incorpora MySQL Administrator y MySQL Query, ambos son gratuitos en su descarga y utilización.

Para el proyecto, usaremos el programa SQL MAESTRO este lo puedes descargar de la dirección: www.sqlmaestro.com.

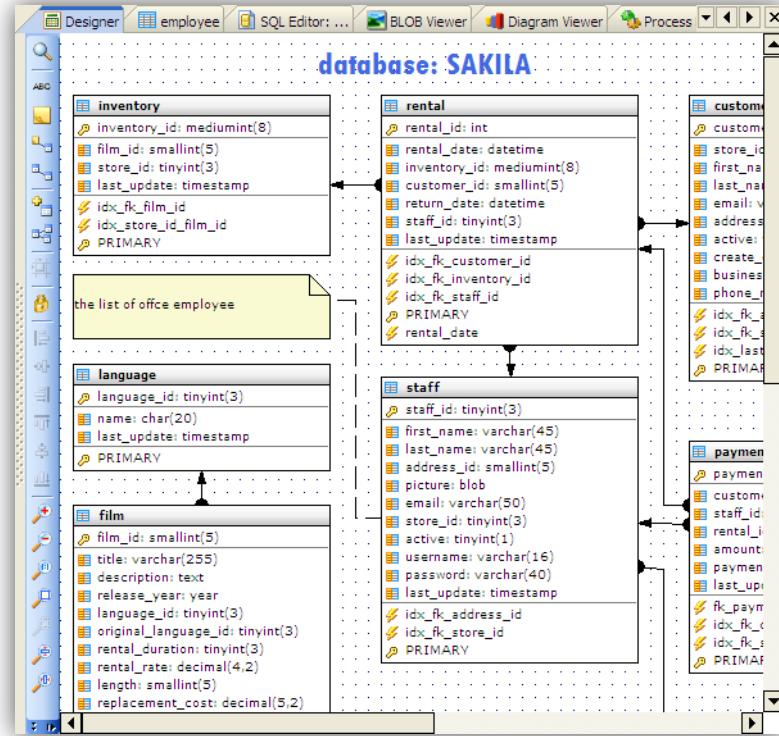
Este administrador permite el acceso a los objetos del manejador sin tener que estar trabajando directamente con la consola o con el phpmyadmin.

Aquí podemos ver y tener acceso a los triggers, procedimientos almacenados y cada uno de los elementos que tiene como vistas, usuarios, tablas, columnas, variables, así como diferentes servidores.

Otra herramienta que existe es el navegador de objetos, una vez que seleccionamos un objeto podemos trabajar con él.



Trabajando con datos podemos obtener los diseñadores, tanto de reportes como de relaciones.



Cada uno de estos elementos nos facilita la realización del proyecto.

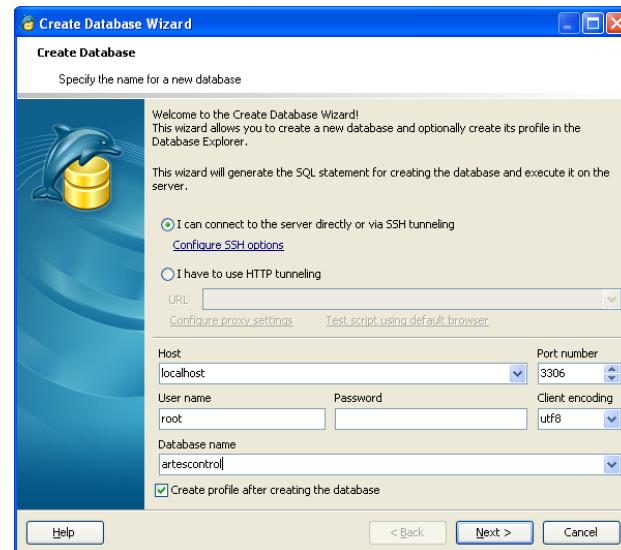
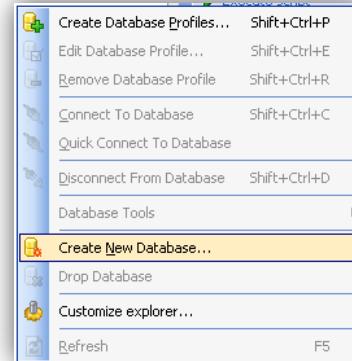
CREANDO BASE DE DATOS

Objetivos

- Instalar SQL Maestro for MySQL
- Crear base de datos
- Crear tablas
- Crear índices y claves foráneas

Para iniciar vamos a agregar una base de datos a nuestro servidor, usando SQL Maestro for Mysql, damos con el clic derecho sobre el explorador de objetos.

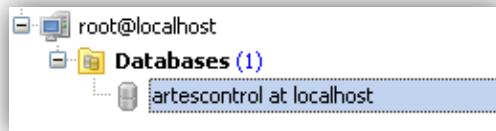
Creamos una base de datos, el asistente nos ira pidiendo datos como lo es el nombre de usuario, la dirección a la que se conectara, el puerto. Previo a esto deberíamos tener instalado el proyecto WAMP e iniciado para tener acceso al manejador.



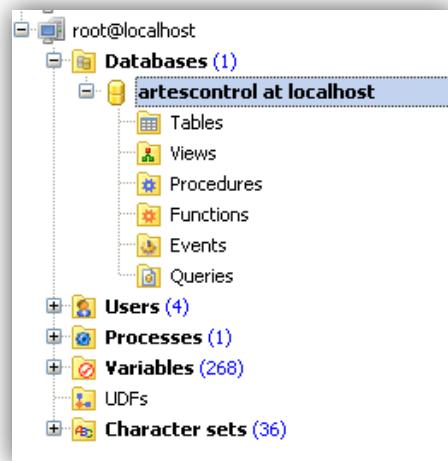
A esta base le podremos el nombre de “artescontrol” después de este asistente al darle clic a ready estaremos en el asistente de conexión para profile, aceptamos las

indicaciones y ahora podemos ver dentro de nuestro explorador de objetos el indicador de que tenemos un servidor registrado.

Si se ha manejado SQL Server Management Studio puede ver una similitud entre estas herramientas.



Una vez que se tiene el profile, lo único es generar un doble clic sobre el nombre del perfil y esperar a que genere todo el explorador de objetos.



Tenemos tablas, vistas, procedimientos, funciones, eventos, consultas, usuarios, procesos, variables, etc. Cada uno de ellos está listo para poder llevar a cabo la tarea para la cual está diseñada.

Partiendo de la necesidad de la institución, veamos que se requiere tener un control de alumnos con ciertos atributos que nos permitirán concentrar la información y explotarla en un futuro. Lo primero que debemos hacer es generar un clic derecho sobre el objeto "Tables". Y seleccionar la opción "Create New Table". El asistente nos pedirá el nombre de la tabla a la cual pondremos antes de la tabla de alumnos, la de usuarios del sistema con los siguientes campos:

usuarios	
id_usuario:	int
nombre:	varchar(60)
sesion:	varchar(30)
acceso:	char(32)
nivel:	int
PRIMARY	

Dejando como clave primaria el **id_usuario**, nuestro explorador de objetos ahora lucirá de esta manera:

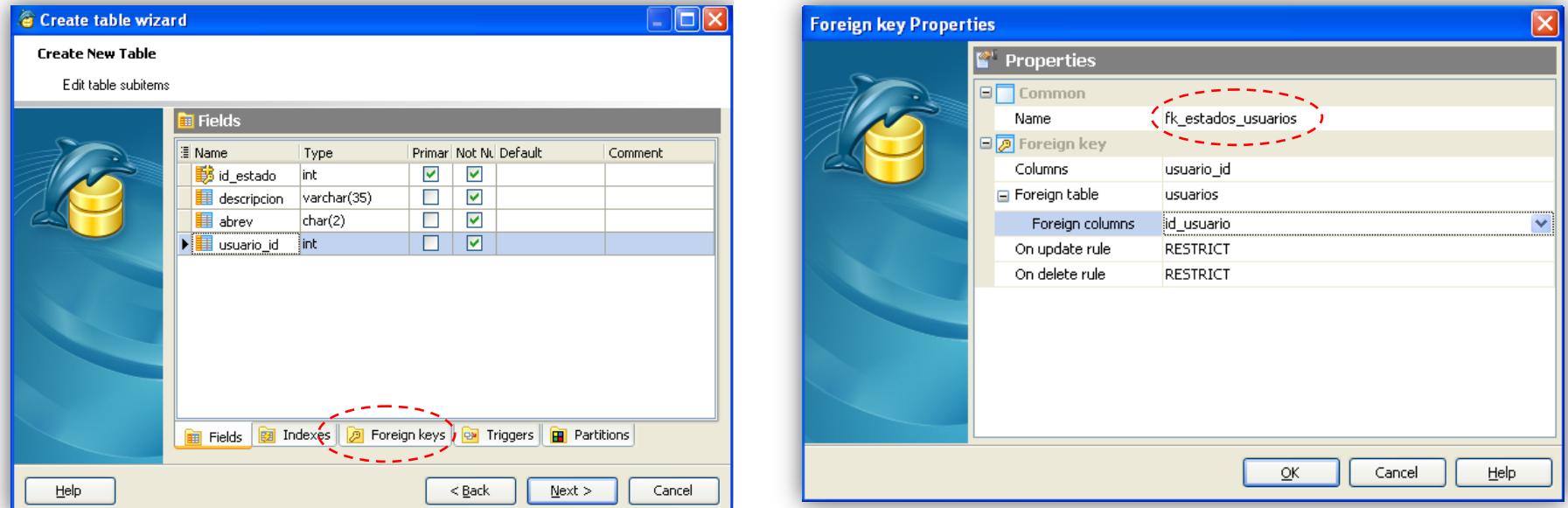
estados	
id_estado: int	
estado: varchar(31)	
abrev: char(2)	
usuario_id: int	
PRIMARY	
fk_estados_usuarios	

municipios	
id_municipio: int	
municipio: varchar(35)	
estados_id: int	
usuarios_id: int	
fk_municipios_estados	
fk_municipios_usuarios	
PRIMARY	

De esta manera trabajaremos con las tablas maestro que no requieren de claves foráneas para poder trabajar.

Para crear una clave foránea es necesario tener el mismo tipo de dato y la cantidad de caracteres a utilizar para generar este tipo de referencia, ahora vamos a crear los estados de la república y una tabla de municipios, teniendo municipios dos claves foráneas, y estado solo una.

Para generar estas claves foráneas es necesario estar en el asistente de creación de tablas y seleccionar foreign key.



Y sobre el area en blanco dar un clic derecho y seleccionar “Add New Foreign Key”, nos va a pedir el nombre del constraint, este lo iniciaremos con fk_nombredetabla_nombredetabladestino. En este caso se llamará “fk_estados_usuarios”.

Y damos clic a OK, una vez terminado, seguimos el asistente, “Next”, “Ready” y “Execute”. Cuando se termine aparece en el lado derecho el detalle del objeto, en este caso el objeto seleccionado es la tabla “estados” y nos permite ver las propiedades, los datos, los permisos y por último el SQL que lo ha generado.

```

Properties Data Permissions SQL
-- Table: estados
-- DROP TABLE IF EXISTS `estados`;

CREATE TABLE `estados` (
    `id_estado` int AUTO_INCREMENT NOT NULL,
    `descripcion` varchar(35) NOT NULL,
    `abrev` char(2) NOT NULL,
    `usuario_id` int NOT NULL,
    /* Keys */
    PRIMARY KEY (`id_estado`),
    /* Foreign keys */
    CONSTRAINT `fk_estados_usuarios`
        FOREIGN KEY (`usuario_id`)
        REFERENCES `usuarios`(`id_usuario`)
        ON DELETE RESTRICT
        ON UPDATE RESTRICT
) ENGINE = InnoDB;

CREATE INDEX `fk_estados_usuarios`
ON `estados`
(`usuario_id`);

```

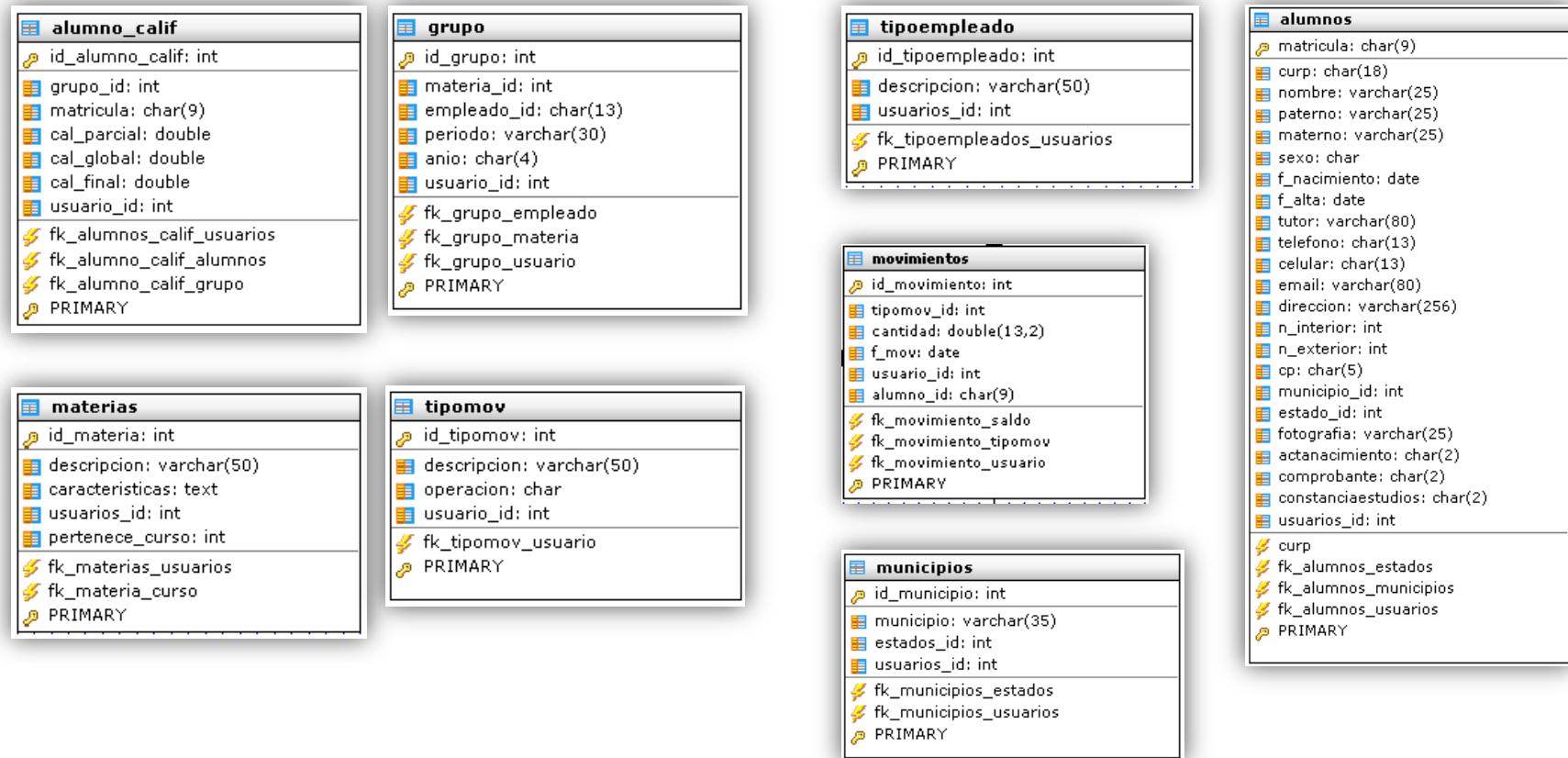
saldo	
alumno_id:	char(9)
saldo:	double(13,2)
usuario_id:	int
fk_saldo_alumno:	
fk_saldo_usuario:	
PRIMARY	

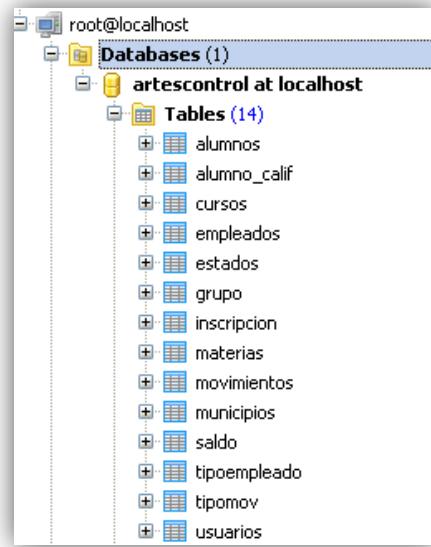
inscripcion	
id_inscripcion:	int
f_inscripcion:	date
semestre:	char
curso_id:	int
alumno_id:	char(9)
usuario_id:	int
fk_inscripcion_alumno:	
fk_inscripcion_curso:	
fk_inscripcion_usuario:	
PRIMARY	

cursos	
id_curso:	int
nombre_curso:	varchar(100)
usuario_id:	int
PRIMARY	
fk_curso_usuario:	

empleados	
rfc:	char(13)
nombre:	varchar(25)
paterno:	varchar(25)
materno:	varchar(25)
direccion:	varchar(80)
n_interior:	int
n_exterior:	int
cp:	char(5)
municipio_id:	int
estado_id:	int
tipoempleado_id:	int
cv:	longtext
foto:	varchar(25)
telefono:	char(13)
celular:	char(13)
email:	varchar(50)
f_nacimiento:	date
usuario_id:	int
comprobante:	char(2)
hmedico:	char(2)
sempleado:	char(2)
fk_empleado_estado:	
fk_empleado_municipio:	
fk_empleado_tipoempleado:	
fk_empleado_usuario:	
PRIMARY	

Recuerden siempre usar el tipo de tabla transaccional, INNODB. Ahora solo queda generar cada una de las tablas restantes con sus respectivas claves foráneas.





Una vez que se termine el esquema quedará de esta manera.

GENERANDO PROCEDIMIENTOS ALMACENADOS

Objetivos

- ▶ Entender cómo afectan los procedimientos
- ▶ Crear procedimientos para cada una de las tablas
- ▶ Crear procedimientos que impacte más de una tabla

Dentro de las base de datos, cuidar la integridad referencia ha sido siempre prioridad, por esta razón es requerido un procedimiento para cada uno de las tablas, ya que ni en PHP es seguro al 100% dejar este tipo de información, pues podemos caer en el error y ser vulnerables a ataques por inyección de SQL.

Los procedimientos almacenados y funciones son nuevas funcionalidades de la versión de MySQL 5.0. Un procedimiento almacenado es un conjunto de comandos SQL que pueden almacenarse en el servidor. Una vez que se hace, los clientes no necesitan relanzar los comandos individuales pero pueden en su lugar referirse al procedimiento almacenado.

Algunas situaciones en que los procedimientos almacenados pueden ser particularmente útiles:

- Cuando múltiples aplicaciones cliente se escriben en distintos lenguajes o funcionan en distintas plataformas, pero necesitan realizar la misma operación en la base de datos.
- Cuando la seguridad es muy importante. Los bancos, por ejemplo, usan procedimientos almacenados para todas las operaciones comunes.

Esto proporciona un entorno seguro y consistente, y los procedimientos pueden asegurar que cada operación se loguea apropiadamente.

En tal entorno, las aplicaciones y los usuarios no obtendrían ningún acceso directo a las tablas de la base de datos, sólo pueden ejecutar algunos procedimientos almacenados.

Los procedimientos almacenados pueden mejorar el rendimiento ya que se necesita enviar menos información entre el servidor y el cliente. El intercambio que hay es que aumenta la carga del servidor de la base de datos ya que la mayoría del trabajo se realiza en la parte del servidor y no en el cliente. Considere esto si muchas máquinas cliente (como servidores Web) se sirven a sólo uno o pocos servidores de bases de datos.

Los procedimientos almacenados le permiten tener bibliotecas o funciones en el servidor de base de datos. Esta característica es compartida por los lenguajes de programación modernos que permiten este diseño interno, por ejemplo, usando clases. Usando estas características del lenguaje de programación cliente es beneficioso para el programador incluso fuera del entorno de la base de datos.

MySQL sigue la sintaxis SQL:2003 para procedimientos almacenados, que también usa IBM DB2.

Procedimientos almacenados y las tablas de permisos

Los procedimientos almacenados requieren la tabla proc en la base de datos mysql. Esta tabla se crea durante la instalación de MySQL 5.0. Si está actualizando a MySQL 5.0 desde una versión anterior, asegúrese de actualizar sus tablas de permisos para asegurar que la tabla proc existe.

Desde MySQL 5.0.3, el sistema de permisos se ha modificado para tener en cuenta los procedimientos almacenados como sigue:

- El permiso CREATE ROUTINE se necesita para crear procedimientos almacenados.
- El permiso ALTER ROUTINE se necesita para alterar o borrar procedimientos almacenados. Este permiso se da automáticamente al creador de una rutina.
- El permiso EXECUTE se requiere para ejecutar procedimientos almacenados. Sin embargo, este permiso se da automáticamente al creador de la rutina. También, la característica SQL SECURITY por defecto para una rutina es DEFINER, lo que permite a los usuarios que tienen acceso a la base de datos ejecutar la rutina asociada.

```
CREATE PROCEDURE sp_name ([parameter[,...]])
[characteristic ...] routine_body

CREATE FUNCTION sp_name ([parameter[,...]])
RETURNS type
[characteristic ...] routine_body

parameter:
[ IN | OUT | INOUT ] param_name type

type:
Any valid MySQL data type

characteristic:
LANGUAGE SQL
| [NOT] DETERMINISTIC
{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
SQL SECURITY { DEFINER | INVOKER }
COMMENT 'string'

routine body:
procedimientos almacenados o comandos SQL válidos
```

Declarar variables locales con DECLARE

Este comando se usa para declarar variables locales. Para proporcionar un valor por defecto para la variable, incluya una cláusula DEFAULT

El valor puede especificarse como expresión, no necesita ser una constante. Si la cláusula DEFAULT no está presente, el valor inicial es NULL.

La visibilidad de una variable local es dentro del bloque BEGIN ... END donde está declarado. Puede usarse en bloques anidados excepto aquéllos que declaren una variable con el mismo nombre.

Sentencia SET para variables

El comando SET en procedimientos almacenados es una versión extendida del comando general SET. Las variables referenciadas pueden ser las declaradas dentro de una rutina, o variables de servidor globales.

El comando SET en procedimientos almacenados se implementa como parte de la sintaxis SET pre-existente. Esto permite una sintaxis extendida de SET a=x, b=y, ... donde distintos tipos de variables (variables declaradas local y globalmente y variables de sesión del servidor) pueden mezclarse. Esto permite combinaciones de variables locales y algunas opciones que tienen sentido sólo para variables de sistema; en tal caso, las opciones se reconocen pero se ignoran.

Sentencia IF

```
IF search_condition THEN statement_list
[ELSEIF search_condition THEN _statement_list] ...
[ELSE statement_list]
END IF
```

IF implementa un constructor condicional básico. Si search_condition se evalúa a cierto, el comando SQL correspondiente listado se ejecuta. Si no coincide ninguna search_condition se ejecuta el comando listado en la cláusula ELSE. statement_list puede consistir en varios comandos.

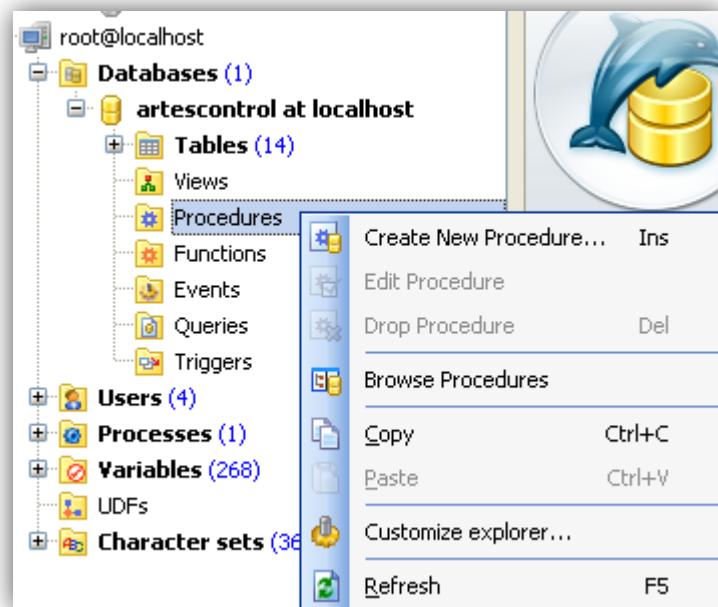
Tenga en cuenta que también hay una función IF(), que difiere del comando IF descrito aquí.

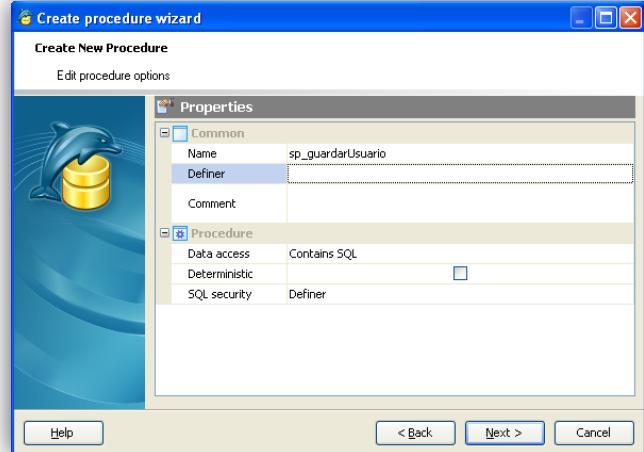
Y seleccionamos el menú “Create New Procedure”. Aparecerá un asistente para crear los procedimientos, el cual nos pide el nombre del procedimiento.

Procedimiento para guardar usuarios del sistema

Para poder realizar este proceso, es necesario entender cómo funcionan los procedimientos visto en los temas anteriores.

Usando el programa SQL Maestro for MySQL, vamos a realizar un nuevo procedimiento, en el explorador de objetos vamos darle clic derecho al contenedor de objetos “Procedures”.





Seguimos el asistente y el nos pedirá ahora los parámetros que este procedimiento tendrá, aquí es donde es necesario colocar los tipos de datos idénticos a los que se encuentran en la tabla o tablas a afectar.

Hasta este punto solo queda seguir con "Next", "Ready", "Execute". Nos solicitará el contenido del procedimiento, aquí es donde se hace todo el proceso, aquí introduciremos la sentencia **INSERT INTO** para afectar a la tabla **usuarios**.

```
Definition
BEGIN
    /* Procedure text */
    INSERT INTO usuarios VALUES ('',nombre,sesion,acceso,nivel);
END
```

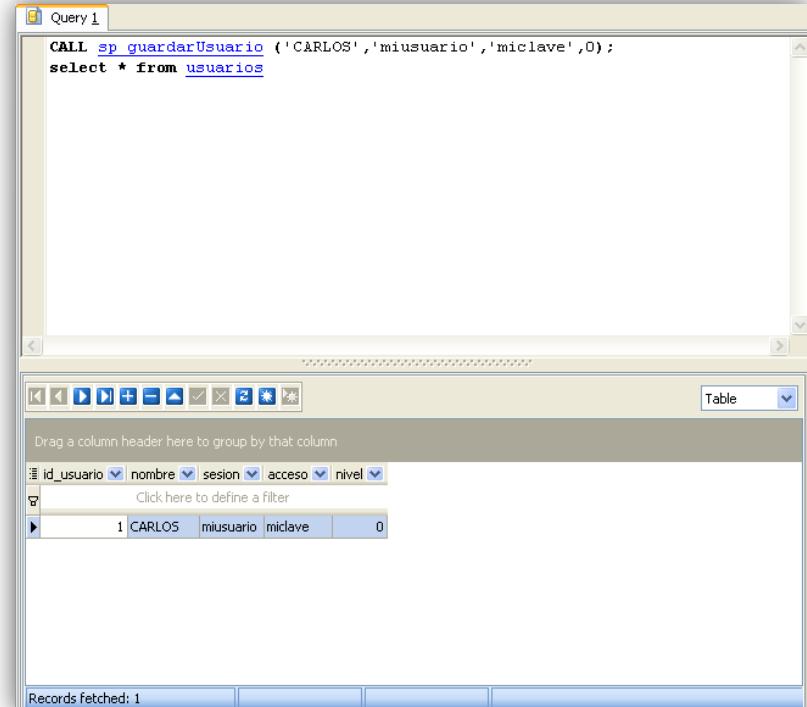
Para llamar el procedimiento sólo hay que hacer uso de la sentencia CALL, dentro de un editor SQL.



```
CALL sp_guardarUsuario ('CARLOS','miusuario','miclave',0);
```

The command(s) completed successfully.

Solo resta verificar si la información se ha guardado correctamente y ahora tenemos a ese usuario listo para ser utilizado.



id_usuario	nombre	sesion	acceso	nivel
1	CARLOS	miusuario	miclave	0

Records fetched: 1

Hasta aquí hemos creado un procedimiento que afecta a la tabla usuarios, comprendido como se crean y modifican. Lo que nos queda es mostrar el uso de DECLARE y SET, los cuales nos permiten crear variables locales dentro del BEGIN y END. La sintaxis es la siguiente

DECLARE variable tipo;

SET variable=valor;

Regresando al procedimiento anterior lo que tenemos que hacer es generar que todos los niveles sean de tipo uno.

```
BEGIN
    /* Procedure text */
    DECLARE miNivel int;
    set miNivel=1;
    INSERT INTO usuarios VALUES ('',nombre,sesion,acceso,miNivel);
END
```

Ya terminado la declaración, podemos prescindir del parámetro “nivel”

Parameters		
Name	Type	Scope
(t) nombre	varchar(60)	Input
(t) sesion	varchar(30)	Input
(t) acceso	char(32)	Input

Parameters		
Name	Type	Scope
(t) nombreP	varchar(60)	Input
(t) sesion	varchar(30)	Input
(t) acceso	char(32)	Input

Definition		
<pre>BEGIN /* Procedure text */ DECLARE miNivel int; set miNivel=1; INSERT INTO usuarios VALUES ('',nombreP,sesion,acceso,miNivel); select * from usuarios where nombre=nombreP; END </pre>		

Filtrando un poco más podemos tener una consulta dentro del procedimiento.

Lo que ahora tenemos que generar será los procedimientos para guardar en cada una de las tablas, iniciamos con “tipoempleado” llamado “sp_guardarTipoEmpleado”

Properties				Results	Permissions	SQL
Parameters				Definition		
Name	Type	Scope				
(t) usuario	int					
(t) descripcion	varchar(50)					

Definition		
<pre>BEGIN /* Procedure text */ INSERT INTO tipoempleado VALUES ('',descripcion,usuario); END </pre>		

Una vez terminado el procedimiento, guardaremos 6 tipos de empleados.

Query 1

```

CALL sp_guardarTipoEmpleado ('DIRECTOR GENERAL',1);
CALL sp_guardarTipoEmpleado ('SUBDIRECTOR',1);
CALL sp_guardarTipoEmpleado ('PROFESOR DE ASIGNATURA',1);
CALL sp_guardarTipoEmpleado ('ADMINISTRADOR',1);
CALL sp_guardarTipoEmpleado ('SECRETARIA',1);
CALL sp_guardarTipoEmpleado ('DIRECTOR DE CONTROL ESCOLAR',1);

SELECT * FROM tipoempleado;

```

Result 1

	id_tipoempleado	descripcion	usuarios_id
▶	13	DIRECTOR GENERAL	1
▶	14	SUBDIRECTOR	1
▶	15	PROFESOR DE ASIGNATURA	1
▶	16	ADMINISTRADOR	1
▶	17	SECRETARIA	1
▶	18	DIRECTOR DE CONTROL ESCOLAR	1

Una vez terminado los tipos de empleados, podemos crear el procedimiento para guardar los estados y municipio, con los nombres “sp_guardarEstado” y “sp_guardarMunicipio”.

Parameters

Name	Type	Scope
(I) id_estado	int	Input
(I) estado	varchar(31)	Input
(I) abrev	char(2)	Input
(I) usuario_id	int	Input

Definition

```

BEGIN
    /* Procedure text */
    INSERT INTO estados VALUES (id_estado,estado,abrev,usuario_id);
END

```

Parameters

Name	Type	Scope
(I) municipio	varchar(35)	Input
(I) estado_id	int	Input
(I) usuario_id	int	Input

Definition

```

BEGIN
    /* Procedure text */
    INSERT INTO municipios VALUES ('',municipio,estado_id,usuario_id);
END

```

Despues de generarlos, estamos listos para agregarlos a los empleados a la base de datos, con el siguiente script “sp_guardarEmpleado”

Parameters		
Name	Type	Scope
(t) rfc	char(13)	Input
(t) nombre	varchar(25)	Input
(t) paterno	varchar(25)	Input
(t) materno	varchar(25)	Input
(t) direccion	varchar(80)	Input
(t) n_interior	int	Input
(t) n_exterior	int	Input
(t) cp	char(5)	Input
(t) municipio_id	int	Input
(t) estado_id	int	Input
(t) tipoempleado_id	int	Input
(t) cv	longtext	Input
(t) foto	varchar(25)	Input
(t) telefono	char(13)	Input
(t) celular	char(13)	Input
(t) email	varchar(50)	Input
(t) f_nacimiento	date	Input
(t) usuario_id	int	Input
(t) comprobante	char(2)	Input
(t) hmedico	char(2)	Input
(t) sempleo	char(2)	Input

Una vez terminado la lista de parámetros, iniciamos con el script de guardado:

```
BEGIN
    /* Procedure text */
    INSERT INTO empleados VALUES (rfc,nombre,paterno,materno,direccion,
        n_interior,n_exterior,cp,municipio_id,estado_id,tipoempleado_id,
        cv,foto,telefono,celular,email,f_nacimiento,usuario_id,comprobante,
        hmedico,sempleo);
END
```

Para poder probarlos, vamos a generar nuestras llamadas a los procedimientos, en orden jerárquico, sería: Estados – Municipios – Empleados.

```
CALL sp_guardarTipoEmpleado ('DIRECTOR GENERAL',1);
CALL sp_guardarTipoEmpleado ('SUBDIRECTOR',1);
CALL sp_guardarTipoEmpleado ('PROFESOR DE ASIGNATURA',1);
CALL sp_guardarTipoEmpleado ('ADMINISTRADOR',1);
CALL sp_guardarTipoEmpleado ('SECRETARIA',1);
CALL sp_guardarTipoEmpleado ('DIRECTOR DE CONTROL ESCOLAR',1);
```

```
CALL sp_guardarEstado (13,'HIDALGO','HG',1);
CALL sp_guardarMunicipio ('PACHUCA DE SOTO',13,1);
```

```
CALL sp_guardarEmpleado('ZAOC810504HG5','CARLA','ZAMORA','ORTEGA',
    'HACIENDA HUEYAPAN',NULL,123,'42000',3,13,13,'POR AHORA ESTA PENDIENTE
    EL CV','default.jpg','7711324654','7716988887','carlazamora@hotmail.com',
    '04/05/1981',1,'SI','NO','SI');
```

Ahora es tiempo de generar los procedimientos de materias y cursos.
“sp_guardarCurso”

Parameters		
Name	Type	Scope
(t) nombre_curso	varchar(100)	Input
(t) usuario_id	int	Input
.....		
Definition		
<pre>BEGIN /* Procedure text */ INSERT INTO cursos VALUES ('',nombre_curso,usuario_id); END</pre>		

“sp_guardarMateria”

Parameters

Name	Type	Scope
(+) descripcion	varchar(50)	Input
(+) caracteristicas	text	Input
(+) usuario_id	int	Input
(+) pertenece_curso	int	Input

Definition

```

BEGIN
    /* Procedure text */
    INSERT INTO materias VALUES('',descripcion,
                                caracteristicas,usuario_id,pertenece_curso);
END

```

```

CALL sp_guardarCurso ('DANZA',1);
CALL sp_guardarMateria ('COMPRESION DEL ARTE','LOS OBJETIVOS DE LA MATERIA
ES MOSTRAR AL ALUMNO LA FACILIDAD CON LA QUE EL CUERPO PUEDE
TENER MEJOR DESEMPEÑO PARTIENDO DE LA COMPRESION DEL ARTE',1,1);

```

“sp_guardarGrupo”

Parameters

Name	Type	Scope
(+) materia_id	int	Input
(+) empleado_id	char(13)	Input
(+) periodo	varchar(30)	Input
(+) anio	char(4)	Input
(+) usuario_id	int	Input

Definition

```

BEGIN
    /* Procedure text */
    INSERT INTO grupo VALUES ('',materia_id,empleado_id,
                            periodo,anio,usuario_id);
END

```

```
CALL sp_guardarGrupo (1,'ZAO810504HG5','ENERO-JUNIO','2009',1);
```

“sp_guardarTipoMovimiento”

The screenshot shows the 'Parameters' section with three input parameters: 'descripcion' (varchar(50)), 'operacion' (char), and 'usuario_id' (int). The 'Definition' section contains the following PL/SQL code:

```
BEGIN
    /* Procedure text */
    INSERT INTO tipomov VALUES ('',descripcion,operacion,usuario_id);
END
```

```
call sp_guardarTipoMovimiento
    ('INICIALIZAR ALUMNO EN EL SISTEMA','N',1);
```

```
call sp_guardarTipoMovimiento
    ('INSCRIPCIÓN','S',1);
call sp_guardarTipoMovimiento
    ('PAGO DE INSCRIPCIÓN','R',1);
call sp_guardarTipoMovimiento
    ('REINSCRIPCIÓN','S',1);
call sp_guardarTipoMovimiento
    ('PAGO DE REINSCRIPCIÓN','R',1);
call sp_guardarTipoMovimiento
    ('HISTORIAL ACADEMICO','S',1);
call sp_guardarTipoMovimiento
    ('PAGO DE HISTORIAL ACADEMICO','R',1);
```

Hasta ahora hemos logrado tener todos los procedimientos bien programados, pero que pasa si requerimos afectar a más de una tabla en el proyecto, como es el caso de MOVIMIENTOS, ALUMNOS, SALDO, INSCRIPCION, ALUMNO_CALIF, estas tablas estarán ligadas no solo por referencia, sino también por disparadores.

Triggers en MySQL

Objetivos

- ▶ Entender como funcionan los disparadores
- ▶ Implementar disparadores en tablas
- ▶ Aplicar disparadores tipo NEW y OLD

A partir de MySQL 5.0.2 se incorporó el soporte básico para disparadores (triggers). Un disparador es un objeto con nombre dentro de una base de datos el cual se asocia con una tabla y se activa cuando ocurre en ésta un evento en particular. Por ejemplo, las siguientes sentencias crean una tabla y un disparador para sentencias INSERT dentro de la tabla. El disparador suma los valores insertados en una de las columnas de la tabla:

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
-> FOR EACH ROW SET @sum = @sum + NEW.amount;
```

Sintaxis de CREATE TRIGGER

```
CREATE TRIGGER nombre_disp momento_disp evento_disp
ON nombre_tabla FOR EACH ROW sentencia_disp
```

Un disparador es un objeto con nombre en una base de datos que se asocia con una tabla, y se activa cuando ocurre un evento en particular para esa tabla.

El disparador queda asociado a la tabla *nombre_tabla*. Esta debe ser una tabla permanente, no puede ser una tabla TEMPORARY ni una vista.

momento_disp es el momento en que el disparador entra en acción. Puede ser BEFORE (antes) o AFTER (después), para indicar que el disparador se ejecute antes o después que la sentencia que lo activa.

evento_disp indica la clase de sentencia que activa al disparador. Puede ser INSERT, UPDATE, o DELETE. Por ejemplo, un disparador BEFORE para sentencias INSERT podría utilizarse para validar los valores a insertar.

No puede haber dos disparadores en una misma tabla que correspondan al mismo momento y sentencia. Por ejemplo, no se pueden tener dos disparadores BEFORE UPDATE. Pero sí es posible tener los disparadores BEFORE UPDATE y BEFORE INSERT o BEFORE UPDATE y AFTER UPDATE.

sentencia_disp es la sentencia que se ejecuta cuando se activa el disparador. Si se desean ejecutar múltiples sentencias, deben colocarse entre BEGIN... END, el constructor de sentencias compuestas. Esto además posibilita emplear las mismas sentencias permitidas en rutinas almacenadas.

```
CREATE TABLE test1(a1 INT);
CREATE TABLE test2(a2 INT);
CREATE TABLE test3(a3 INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
CREATE TABLE test4(
    a4 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    b4 INT DEFAULT 0
);

DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON test1
FOR EACH ROW BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
END

DELIMITER ;

INSERT INTO test3 (a3) VALUES
(NULL), (NULL), (NULL), (NULL), (NULL),
(NULL), (NULL), (NULL), (NULL);

INSERT INTO test4 (a4) VALUES
(0), (0), (0), (0), (0), (0), (0), (0);
```

Las palabras clave OLD y NEW permiten acceder a columnas en los registros afectados por un disparador. (OLD y NEW no son sensibles a mayúsculas). En un

disparador para `INSERT`, solamente puede utilizarse `NEW.nom_col`; ya que no hay una versión anterior del registro.

En un disparador para `DELETE` sólo puede emplearse `OLD.nom_col`, porque no hay un nuevo registro. En un disparador para `UPDATE` se puede emplear `OLD.nom_col` para referirse a las columnas de un registro antes de que sea actualizado, y `NEW.nom_col` para referirse a las columnas del registro luego de actualizarlo.

Una columna precedida por `OLD` es de sólo lectura. Es posible hacer referencia a ella pero no modificarla. Una columna precedida por `NEW` puede ser referenciada si se tiene el privilegio `SELECT` sobre ella. En un disparador `BEFORE`, también es posible cambiar su valor con `SET NEW.nombre_col = valor` si se tiene el privilegio de `UPDATE` sobre ella. Esto significa que un disparador puede usarse para modificar los valores antes que se inserten en un nuevo registro o se empleen para actualizar uno existente.

En un disparador `BEFORE`, el valor de `NEW` para una columna `AUTO_INCREMENT` es 0, no el número secuencial que se generará en forma automática cuando el registro sea realmente insertado. `OLD` y `NEW` son extensiones de MySQL para los disparadores.

MySQL gestiona los errores ocurridos durante la ejecución de disparadores de esta manera:

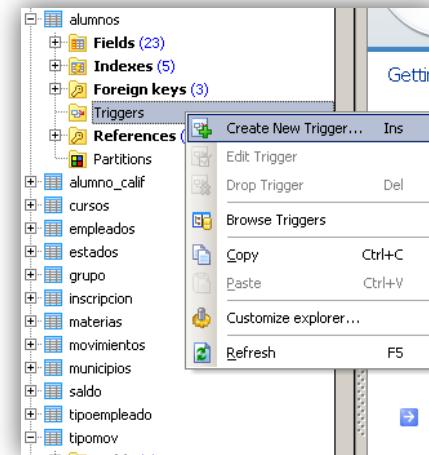
- Si lo que falla es un disparador `BEFORE`, no se ejecuta la operación en el correspondiente registro.
- Un disparador `AFTER` se ejecuta solamente si el disparador `BEFORE` (de existir) y la operación se ejecutaron exitosamente.
- Un error durante la ejecución de un disparador `BEFORE` o `AFTER` deriva en la falla de toda la sentencia que provocó la invocación del disparador.
- En tablas transaccionales, la falla de un disparador (y por lo tanto de toda la sentencia) debería causar la cancelación (rollback) de todos los cambios realizados

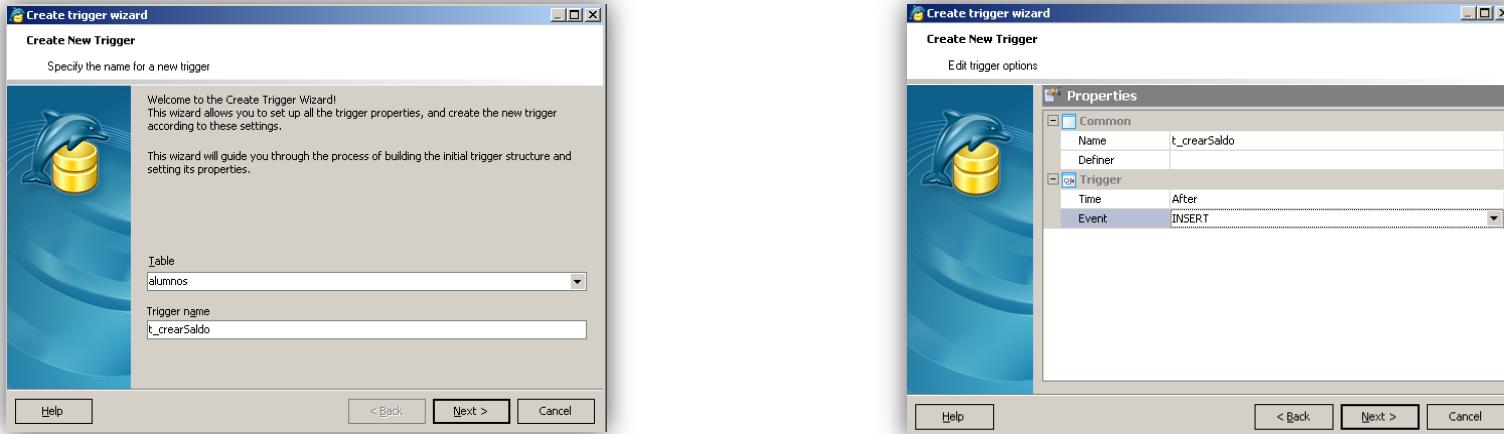
por esa sentencia. En tablas no transaccionales, cualquier cambio realizado antes del error no se ve afectado.

Creando Trigger's para nuestras tablas

Dentro de las tablas a utilizar en el proyecto las que se verán afectadas por un disparador son es la tabla saldo desde inscripción y movimientos.

Creamos el trigger de insert para alumnos que afecte la tabla saldo. Para poder iniciar tenemos que encontrar la tabla sobre la cual vamos a crear el disparador en este caso es “alumnos” en nuestro explorador de objetos desplegamos el contenido y encontramos la carpeta Triggers, le damos con el botón derecho y usamos la opción “Create New Trigger”.





A este trigger lo llamaremos “t_crearSaldo” el cual será disparado después de insertar al alumno.

Cuando nos pida la definición colocaremos el siguiente código:

```
BEGIN
    /* Trigger text */
    INSERT INTO saldo VALUES
        (NEW.matricula,0.00,NEW.usuarios_id);
END
```

Recordando que el operador NEW es quien nos dará la referencia hacia la tabla donde se inserto. Ahora vamos a crear el procedimiento para guardar al alumno, recordando que la tabla de saldo ya está siendo actualizada en automático. “sp_guardarAlumno”

Parameters		
Name	Type	Scope
(#) matricula	char(9)	Input
(#) curp	char(18)	Input
(#) nombre	varchar(25)	Input
(#) paterno	varchar(25)	Input
(#) materno	varchar(25)	Input
(#) sexo	char	Input
(#) f_nacimiento	date	Input
(#) tutor	varchar(80)	Input
(#) telefono	char(13)	Input
(#) celular	char(13)	Input
(#) email	varchar(80)	Input
(#) direccion	varchar(256)	Input
(#) n_interior	int	Input
(#) n_exterior	int	Input
(#) cp	char(5)	Input
(#) municipio_id	int	Input
(#) estado_id	int	Input
(#) fotografía	varchar(25)	Input
(#) actanacimiento	char(2)	Input
(#) comprobante	char(2)	Input
(#) constanciaestudios	char(2)	Input
► (#) usuarios_id	int	Input

```

BEGIN
    /* Procedure text */
    INSERT INTO alumnos VALUES (matricula,
        curp,nombre,paterno,materno,
        sexo,f_nacimiento,now(),tutor,telefono,
        celular,email,direccion,n_interior,
        n_exterior,cp,municipio_id,estado_id,
        fotografía,actanacimiento,comprobante,
        constanciaestudios,usuarios_id);

END

```

```

CALL sp_guardarAlumno ('2009DZ001','GAGA040429MHGRNA7','ANAHI',
    'GARCIA','GARCIA','M','29/04/2004','RAUL GARCIA ESTRADA',
    '5841231245','5846998555','RAULGARCIA@HOTMAIL.COM','LUIS PONCE',
    NULL,768,'98541',3,13,'default.jpg','SI','SI','SI',1);

Select * from alumnos;
Select * from saldo;

```

Ahora podemos ver que el procedimiento esta funcionando y se esta generando el **insert** después de que se guarda el alumno en la tabla saldo, con una deuda inicial de 0.00 pesos.

Lo que haremos ahora es impactar a la tabla de saldo cuando se haga un inserte en la tabla de movimientos. Para esto generaremos en la tabla movimientos un trigger que se llame en after para insert.

```

Definition
BEGIN
    /* Trigger text */
    DECLARE tipo char(1);
    set tipo:= (select operacion from tipomov where id_tipomov=new.tipomov_id);
    IF(tipo='S')THEN
        UPDATE saldo SET saldos= saldos+new.cantidad where alumno_id=new.alumno_id;
    END IF;
    IF(tipo='R') THEN
        UPDATE saldo SET saldos = saldos-new.cantidad where alumno_id=new.alumno_id;
    END IF;
END

```

En esta declaración, estamos usando una variable de tipo char(1), la cual nos va a permitir asignar el resultado del select sobre tipomov, con la condicional de que el id_tipomov sea igual al insertado en la tabla movimiento.

Una vez compilado, ahora crearemos el procedimiento para guardar los movimientos con el nombre “sp_guardarMovimiento”

Parameters

Name	Type	Scope
tipomov_id	int	Input
cantidad	double	Input
usuario_id	int	Input
alumno_id	char(9)	Input

Definition

```

BEGIN
    /* Procedure text */
    INSERT INTO movimientos VALUES
        ('',tipomov_id,cantidad,now(),usuario_id,alumno_id);
END

```

Nos recibirá cuatro parámetros, pero insertaremos 6.

```

Select * from saldo;
select * from movimientos;
call sp_guardarmovimiento (4,500,1,'2009DZ001');
call sp_guardarmovimiento (5,500,1,'2009DZ001');

```

Drag a column header here to group by that column

id_movimiento	tipomov_id	cantidad	f_mov	usuario_id	alumno_id		
	1	4	500	19/02/2009	1	2009DZ001	
		2	5	500	19/02/2009	1	2009DZ001

Al guardar podemos ver los insert sobre la tabla movimientos y veremos también que se actualiza el saldo.

Hasta este momento ya tenemos la mayor parte de nuestra base de datos, ahora estaremos trabajando con la parte de inscripción, la función de esta será guardar a los alumnos a la tabla inscripciones e impactar la lista de alumno_calif, para saber cuáles son las materias que tendrán asignadas, con la limitante de que estén activas y de que ya hayan sido asignadas a un profesor para tener grupo.

Creamos el procedimiento para guardar en inscripción llamado "sp_guardarInscripcion"

Parameters

Name	Type	Scope
semestre	char	Input
curso_id	int	Input
alumno_id	char(9)	Input
usuario_id	int	Input

Definition

```

BEGIN
    /* Procedure text */
    INSERT INTO inscripcion VALUES ('',NOW(),semestre,
                                    curso_id,alumno_id,usuario_id);
END

```

Una vez terminado el procedimiento y antes de utilizarlo, vamos a crear el Trigger que nos permitirá crear la carga de materias de los alumnos con los parámetros AFTER e INSERT sobre la tabla inscripción.

```

BEGIN
    /* Trigger text */
    INSERT INTO alumno_calif SELECT '',g.id_grupo,
    NEW.alumno_id,0,0,0,NEW.usuario_id FROM grupo g
    left join materias m on g.materia_id = m.id_materia
    where m.pertenece_curso=NEW.curso_id and m.semestre = NEW.semestre;
END

```

Esto asignara las materias a los alumnos en la tabla “alumno_calif” y colocará las calificaciones a cero, ya teniendo todo completo probamos el procedimiento con el trigger.

call [sp_guardarInscripcion](#) ('1',1,'2009D2001',1);

id_alumno_calif	grupo_id	matricula	cal_parcial	cal_global	cal_final	usuario_id
1	8	2009D2001	0	0	0	1
2	10	2009D2001	0	0	0	1

Recordemos que la asignación esta en función de los grupos ya creados, esto significa que aunque la carrera y semestre tengan 10 materias pero solo dos estén asignadas para ser impartidas esas son las que se guardarán. Por esta razón es importante ya tener la plantilla del personal antes de hacer la inscripción.

Hasta este punto nuestra base de datos han sido aplicados las características de integridad referencias con las Foreign Key, hemos cuidado el proceso de guardado con los procedimientos e implementando los disparadores dentro de las tablas. A continuación entraremos en el tema de vistas, que serán muy útiles dentro de nuestro sistema.

VISTAS EN MYSQL

Objetivos

- ▶ Conocer la terminología para crear una vista
- ▶ Implementar vistas en nuestra base de datos

Las vistas (incluyendo vistas actualizables) fueron introducidas en la versión 5.0 del servidor de base de datos MySQL. En este capítulo se tratan los siguientes temas:

- Creación o modificación de vistas con `CREATE VIEW` o `ALTER VIEW`
- Eliminación de vistas con `DROP VIEW`
- Obtención de información de definición de una vista (metadatos) con `SHOW CREATE VIEW`

Sintaxis de `CREATE VIEW`

```
CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]  
VIEW nombre_vista [(columnas)]  
AS sentencia_select  
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

Esta sentencia crea una vista nueva o reemplaza una existente si se incluye la cláusula `OR REPLACE`. La `sentencia_select` es una sentencia `SELECT` que proporciona la definición de la vista. Puede estar dirigida a tablas de la base o a otras vistas.

Se requiere que posea el permiso `CREATE VIEW` para la vista, y algún privilegio en cada columna seleccionada por la sentencia `SELECT`.

Para columnas incluidas en otra parte de la sentencia `SELECT` debe poseer el privilegio `SELECT`. Si está presente la cláusula `OR REPLACE`, también deberá tenerse el privilegio `DELETE` para la vista.

Toda vista pertenece a una base de datos. Por defecto, las vistas se crean en la base de datos actual. Para crear una vista en una base de datos específica, indíquela con `base_de_datos.nombre_vista` al momento de crearla.

Las tablas y las vistas comparten el mismo espacio de nombres en la base de datos, por eso, una base de datos no puede contener una tabla y una vista con el mismo nombre.

Al igual que las tablas, las vistas no pueden tener nombres de columnas duplicados. Por defecto, los nombres de las columnas devueltos por la sentencia `SELECT` se usan para las columnas de la vista. Para dar explícitamente un nombre a las columnas de la vista utilice la cláusula `columnas` para indicar una lista de nombres separados con comas. La cantidad de nombres indicados en `columnas` debe ser igual a la cantidad de columnas devueltas por la sentencia `SELECT`.

Las columnas devueltas por la sentencia `SELECT` pueden ser simples referencias a columnas de la tabla, pero tambien pueden ser expresiones contenido funciones, constantes, operadores, etc.

Los nombres de tablas o vistas sin calificar en la sentencia `SELECT` se interpretan como pertenecientes a la base de datos actual. Una vista puede hacer referencia a tablas o vistas en otras bases de datos precediendo el nombre de la tabla o vista con el nombre de la base de datos apropiada.

Las vistas pueden crearse a partir de varios tipos de sentencias `SELECT`. Pueden hacer referencia a tablas o a otras vistas. Pueden usar combinaciones, `UNION`, y subconsultas. El `SELECT` inclusive no necesita hacer referencia a otras tablas. En el siguiente ejemplo se define una vista que selecciona dos columnas de otra tabla, así como una expresión calculada a partir de ellas:

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+---+---+---+
| qty | price | value |
+---+---+---+
|   3 |    50 |   150 |
+---+---+---+
```

La definición de una vista está sujeta a las siguientes limitaciones:

- La sentencia `SELECT` no puede contener una subconsulta en su cláusula `FROM`.
- La sentencia `SELECT` no puede hacer referencia a variables del sistema o del usuario.
- La sentencia `SELECT` no puede hacer referencia a parámetros de sentencia preparados.
- Dentro de una rutina almacenada, la definición no puede hacer referencia a parámetros de la rutina o a variables locales.
- Cualquier tabla o vista referenciada por la definición debe existir. Sin embargo, es posible que después de crear una vista, se elimine alguna tabla o vista a la que se hace referencia. Para comprobar la definición de una vista en busca de problemas de este tipo, utilice la sentencia `CHECK TABLE`.
- La definición no puede hacer referencia a una tabla `TEMPORARY`, y tampoco se puede crear una vista `TEMPORARY`.
- Las tablas mencionadas en la definición de la vista deben existir siempre.
- No se puede asociar un disparador con una vista.

En la definición de una vista está permitido `ORDER BY`, pero es ignorado si se seleccionan columnas de una vista que tiene su propio `ORDER BY`.

Con respecto a otras opciones o cláusulas incluidas en la definición, las mismas se agregan a las opciones o cláusulas de cualquier sentencia que haga referencia a la vista creada, pero el efecto es indefinido. Por ejemplo, si la definición de una vista incluye una cláusula `LIMIT`, y se hace una selección desde la vista utilizando una sentencia que tiene su propia cláusula `LIMIT`, no está definido cuál se aplicará. El mismo principio se extiende a otras opciones como `ALL`, `DISTINCT`, o `SQL_SMALL_RESULT` que se ubican a continuación de la palabra reservada `SELECT`, y a cláusulas como `INTO`, `FOR UPDATE`, `LOCK IN SHARE MODE`, y `PROCEDURE`.

Si se crea una vista y luego se modifica el entorno de proceso de la consulta a través de la modificación de variables del sistema, puede afectar los resultados devueltos por la vista:

```
mysql> CREATE VIEW v AS SELECT CHARSET(CHAR(65)), COLLATION(CHAR(65));
Query OK, 0 rows affected (0.00 sec)

mysql> SET NAMES 'latin1';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM v;
+-----+-----+
| CHARSET(CHAR(65)) | COLLATION(CHAR(65)) |
+-----+-----+
| latin1           | latin1_swedish_ci |
+-----+-----+
1 row in set (0.00 sec)

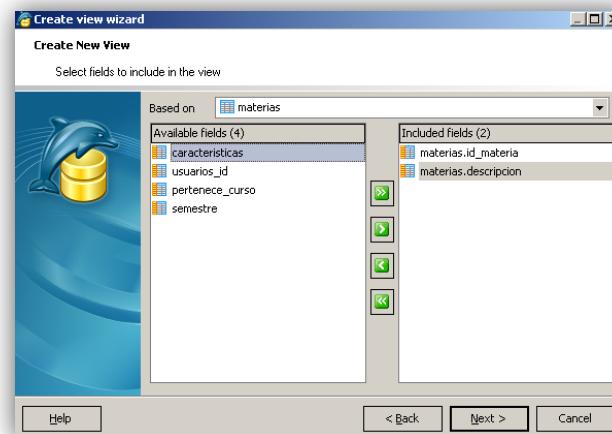
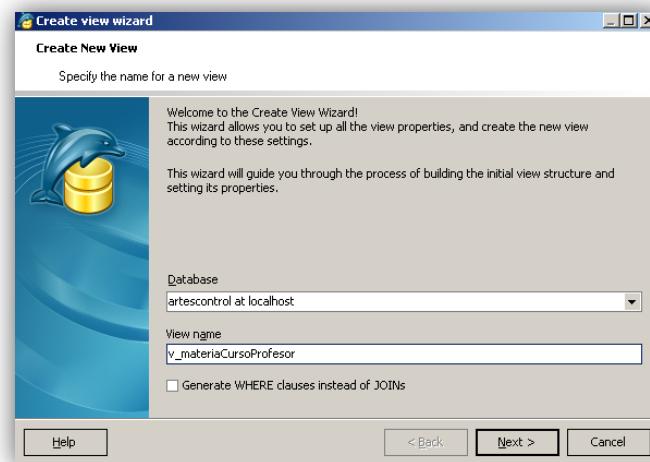
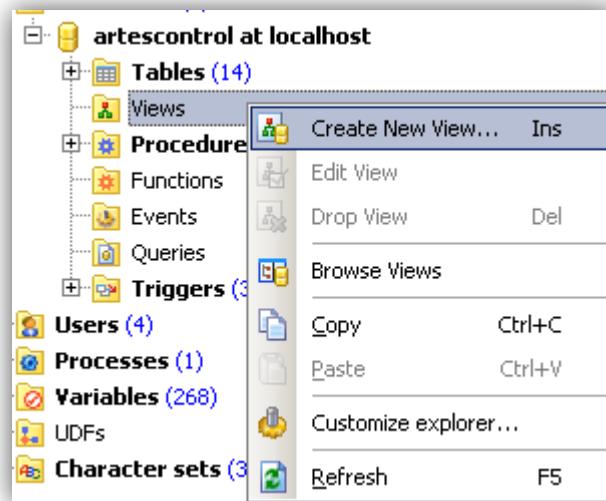
mysql> SET NAMES 'utf8';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM v;
+-----+-----+
| CHARSET(CHAR(65)) | COLLATION(CHAR(65)) |
+-----+-----+
| utf8             | utf8_general_ci  |
+-----+-----+
```

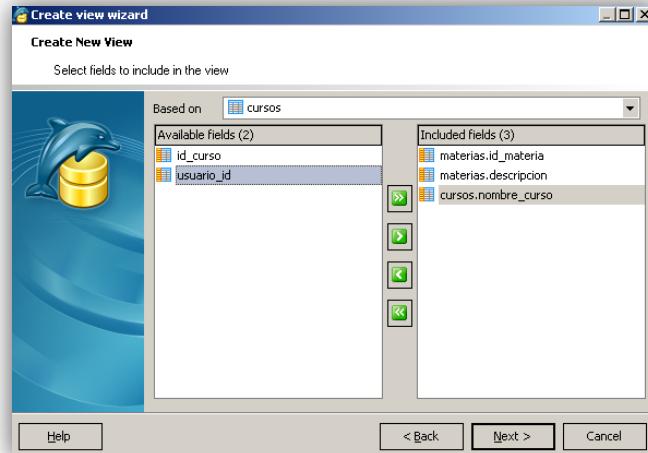
Creando nuestras vistas para el sistema

Dentro del sistema de información que estamos manejando, requerimos de vistas básicas que nos ayudaran a saber cierta información, como la lista de alumnos para el grupo, o bien la lista de materias con el profesor que la imparte. Tal vez requerimos un reporte de todos los alumnos que deben dinero y cuanto deben. Por ahora iniciaremos con una vista sencilla.

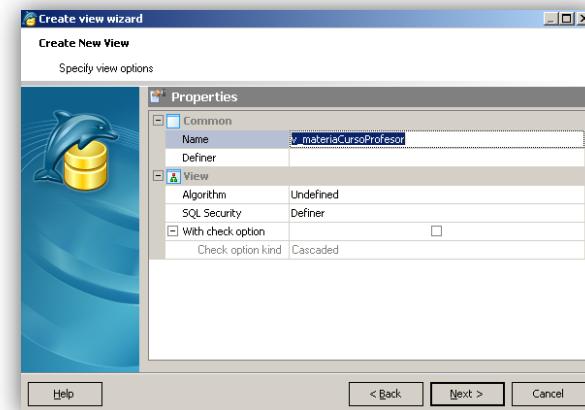
Para crear una vista lo que necesitamos hacer es darle clic derecho al objeto VIEWS



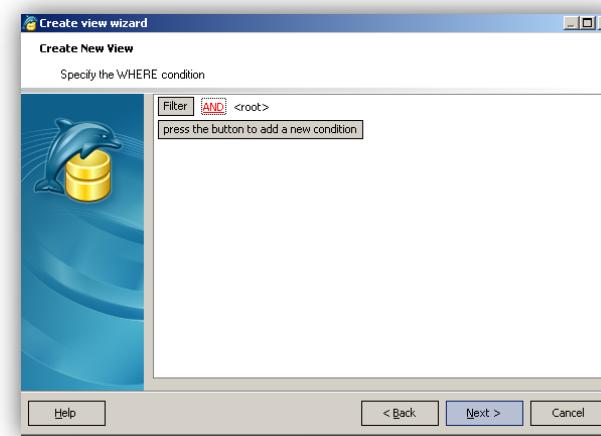
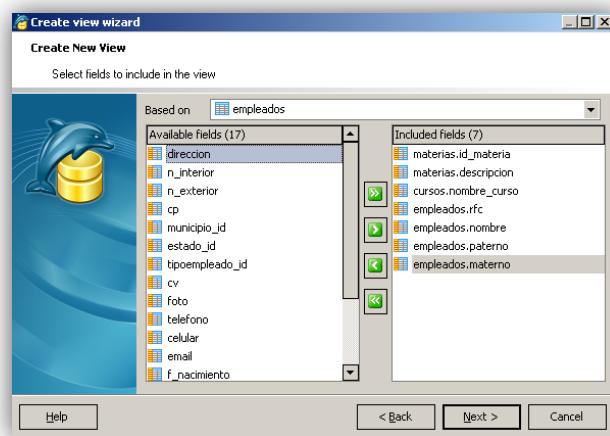
Seleccionamos de la tabla materias los campos id_materia y descripción.



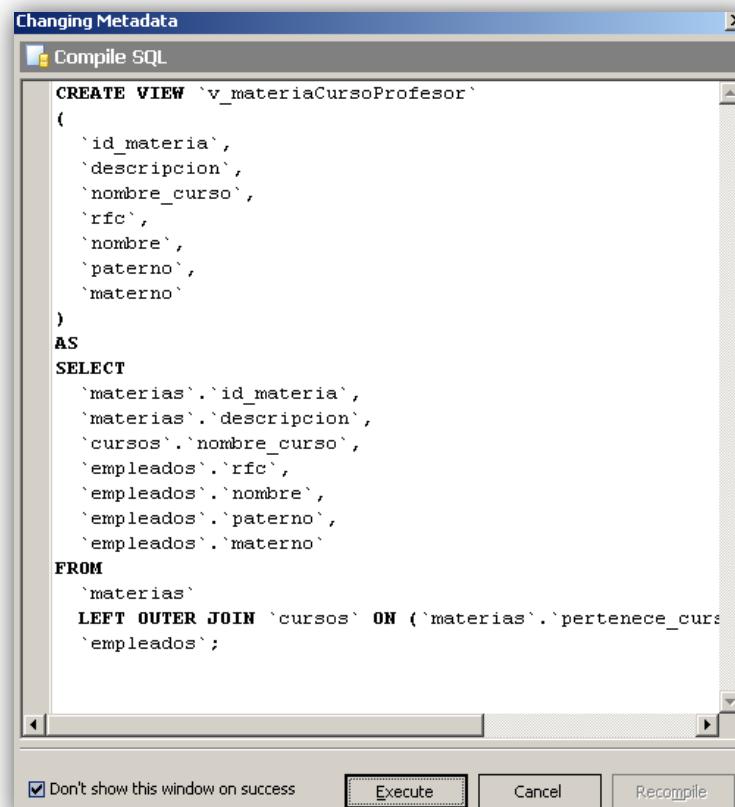
De la tabla empleados el rfc, nombre, paterno y materno, para saber quién es el responsable de cada materia en el curso.



De la tabla curso, el nombre del curso al que pertenece.



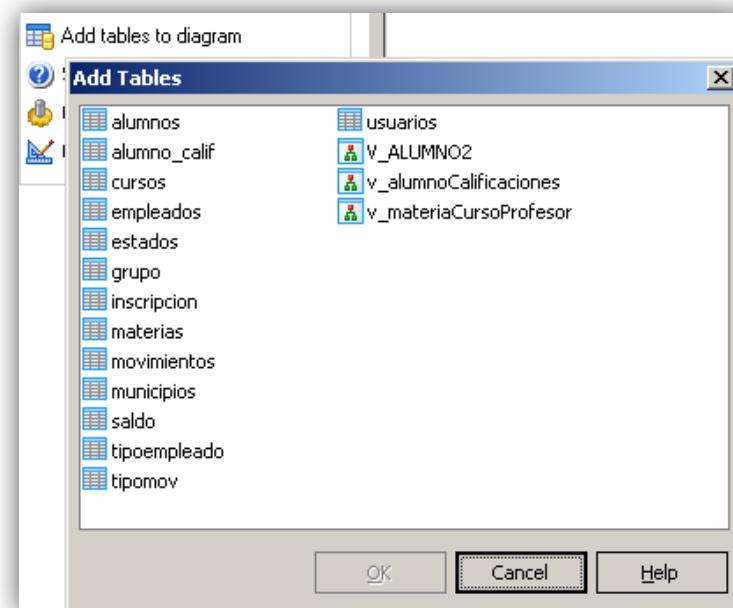
En la imagen anterior podemos ver una lista de condicionales que nos permite el asistente para colocar filtros personalizados sobre cada una de las tablas que se están incluyendo en la vista, en esta ocasión, no la requerimos.



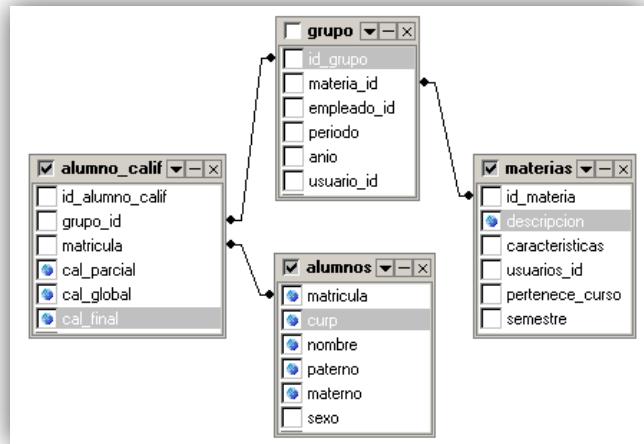
Solo terminamos el asistente con Execute y nos permitirá tener un asistente como para las tablas, para así recorrer los datos que genera la vista.

1	COMPRESION DEL ARTE	DANZA	ZAOC810504HG5	CARLA	ZAMORA	ORTEGA
2	ESCALADA	DANZA	ZAOC810504HG5	CARLA	ZAMORA	ORTEGA
3	ACONDICIONAMIENTO	DANZA	ZAOC810504HG5	CARLA	ZAMORA	ORTEGA
4	LECTURA DE MOVIMIENTOS	DANZA	ZAOC810504HG5	CARLA	ZAMORA	ORTEGA

Otra manera de crear las vistas es con la ayuda del Visual Query Builder,



Aquí vamos a agregar las tablas a utilizar, estas serán las que crearan el cuerpo de nuestra vista.



En la pestaña “editor” podemos ver la consulta creada, esta la podemos seleccionar y copiar, para crear nuestra vista solo tenemos que usar el asistente y no seleccionar ningún campo, solamente tenemos que pegar el código siguiente en el Body de la consulta.

```
Diagram Editor Result
SELECT
    alumnos.matricula,
    alumnos.nombre,
    alumnos.paterno,
    alumnos.materno,
    alumnos.curp,
    materias.descripcion,
    alumno_calif.cal_parcial,
    alumno_calif.cal_global,
    alumno_calif.cal_final
FROM
    alumno_calif
INNER JOIN alumnos ON (alumno_calif.matricula=alumnos.matricula)
INNER JOIN grupo ON (alumno_calif.grupo_id=grupo.id_grupo)
INNER JOIN materias ON (grupo.materia_id=materias.id_materia)
```

Y por ultimo compilamos la vista “v_alumnoCalificaciones” con esto podemos crear la lista de alumnos, si requerimos un filtro, podemos seleccionar por grupo.

```
-- View: v_alumnoCalificaciones
-- DROP VIEW IF EXISTS `v_alumnoCalificaciones`;
CREATE DEFINER = 'root'@'localhost' VIEW `v_alumnoCalificaciones`
(
    `curp`,
    `nombre`,
    `paterno`,
    `descripción`,
    `cal_parcial`,
    `cal_global`,
    `cal_final`
)
AS
select
    `alumnos`.`curp` AS `curp`,
    `alumnos`.`nombre` AS `nombre`,
    `alumnos`.`paterno` AS `paterno`,
    `materias`.`descripción` AS `descripción`,
    `alumno_calif`.`cal_parcial` AS `cal_parcial`,
    `alumno_calif`.`cal_global` AS `cal_global`,
    `alumno_calif`.`cal_final` AS `cal_final`
from (((`alumno_calif`)
join `alumnos` on
(
    (`alumno_calif`).`matricula` = `alumnos`.`matricula`)
))
join `grupo` on
(
    (`alumno_calif`).`grupo_id` = `grupo`.`id_grupo`)
))
join `materias` on
(
    (`grupo`.`materia_id` = `materias`.`id_materia`)
));
```

Este sería el cuerpo de nuestra vista, para consultar a los alumnos y sus calificaciones con materias.

Hasta este momento, no hay necesidad de más vistas, lo que nos resta es llenar nuestros datos para poder acceder al contenido y hacer pruebas del todo nuestra base de datos.

DISEÑO DE LA INTERFAZ CON ADOBE FLEX 3

MODULO 2

OBJETIVOS

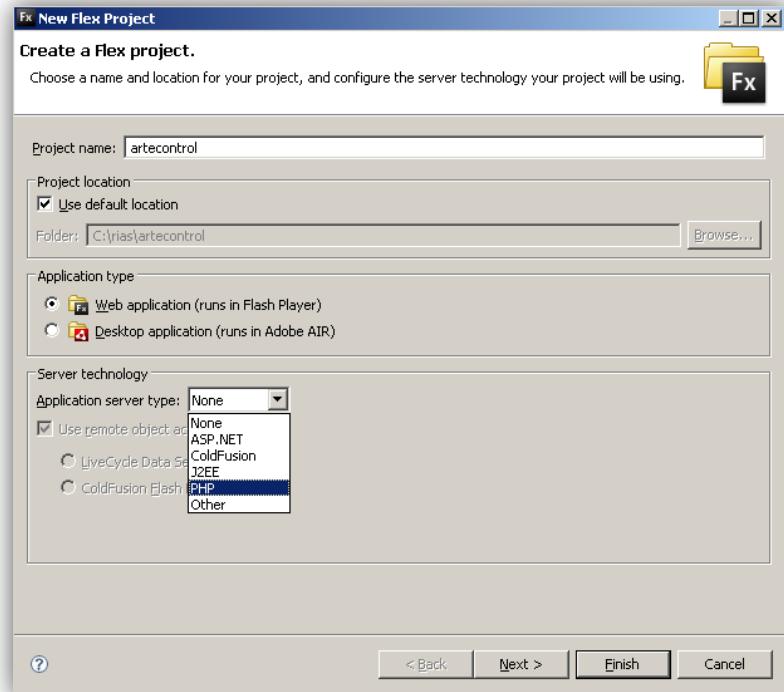
- ▶ Crear un proyecto Flex par web
- ▶ Configurar nuestra carpeta de bin-debug y release
- ▶ Crear el entorno para verificar nuestro usuario.(loggin)
- ▶ Crear menú principal para administrador (estados)

Flex es un marco de trabajo de código abierto gratuito altamente productivo para la creación y el mantenimiento de aplicaciones web expresivas que se implantan coherentemente en los principales exploradores, equipos de sobremesa y sistemas operativos.

Configurando nuestro proyecto

Lo primero que haremos será crear un nuevo proyecto Flex con opción de tecnología del lado del servidor PHP y creando la configuración con WAMP.

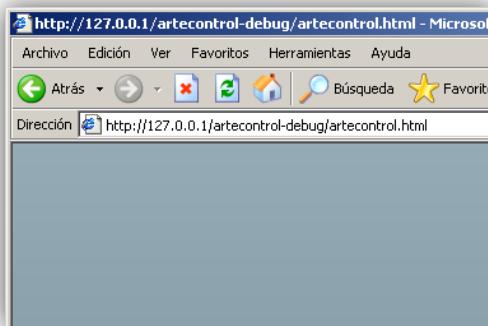
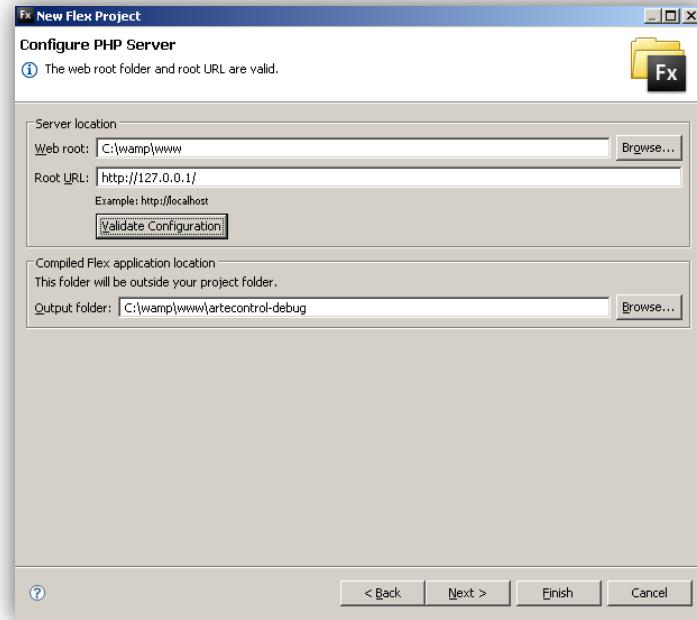
Para esto vamos al menú FILE>NEW>Flex Project y colocamos el nombre “artecontrol” con la opción en Server technology> Application server type = PHP.



Ya que tenemos configurado el proyecto vamos a la opción Next y colocamos los datos en la configuración del servidor PHP.

- ▶ Web root: C:\wamp\www
- ▶ Root Url: <http://127.0.0.1/>

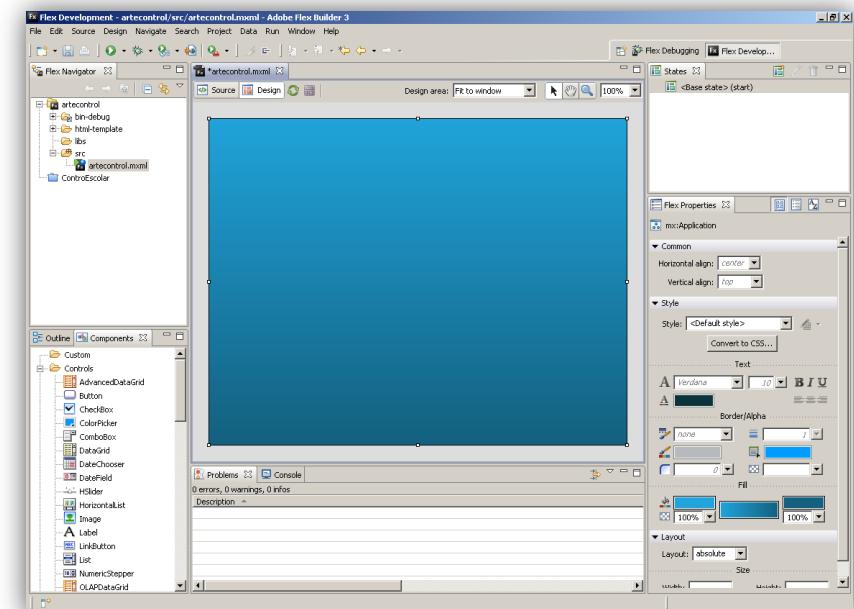
Ya que se tiene esta configuración vamos a validarla con el botón “Validate Configuration”. Y damos un clic al botón “Finish”. Para probar el proyecto solo tenemos que corre con el macro Ctrl + F11 o bien el botón de RUN .



Este sería el resultado de que todo está funcionando correctamente.

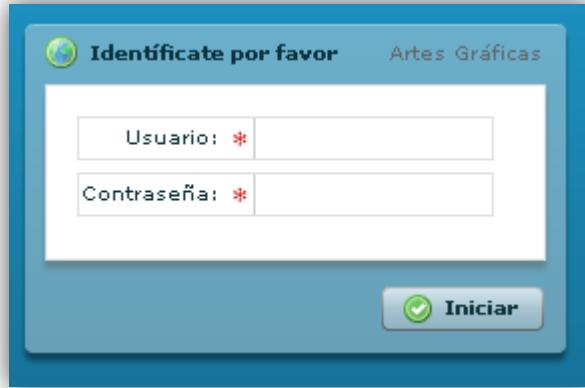
Ya que tenemos nuestro proyecto, vamos a la parte de diseño para empezar a configurar nuestra interfaz de entrada, lo primero será generar un estilo para nuestras background de color azul.

```
backgroundGradientAlphas="[1.0, 1.0]" backgroundGradientColors="#20A4DB, #14607F"
pageTitle="ARTES GRAFICAS DE TIJUANA A.C. ">
```

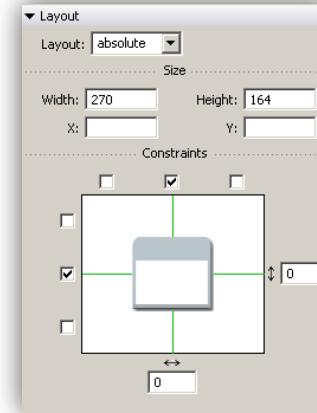


Ahora crearemos nuestra forma para solicitar los datos de firmado a la aplicación, esto lo haremos utilizando:

- Panel
- Form
- ControlBar
- Button
- TextInput

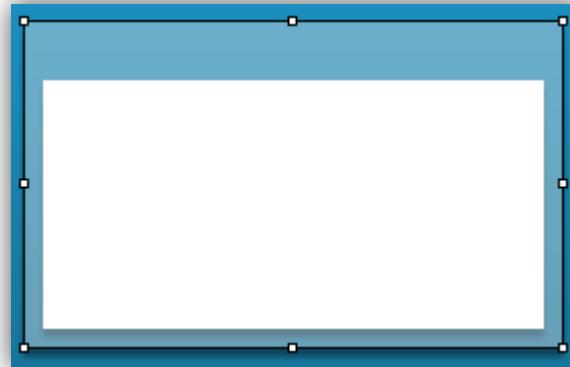


Esta es la manera que debe quedar la forma para poder firmarse en la aplicación, para esto, es necesario cambiar los siguientes atributos en la aplicación. Dentro de la lista de componentes vamos a agregar un control “Panel”, este nos permitirá generar una forma de acceso. Con un tamaño de Width=270, Height=164 con constraints al centro, tanto vertical como horizontal con valores a cero.

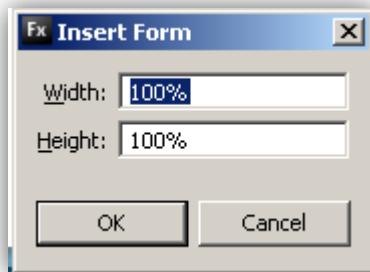


En las propiedades del panel en la sección A-Z buscaremos los siguientes cambios:

- ▶ Status:Artes Gráficas
- ▶ Title: Identificate por favor
- ▶ titleIcon: @Embed(source='assets/iconos/world.png')



Es tiempo de agregar el control “Form” y “ControlBar” así como un Button en el ControlBar alineado a la derecha.



Dejamos caer nuestro control Button dentro del ControlBar y cambiamos el texto a “Iniciar”, también el ICON lo cambiamos

- ▶ Icon: @Embed(source='assets/iconos/accept.png')
- ▶ Label: Iniciar
- ▶ ID: cmdIdentificar

Boton Quedara de esta manera:

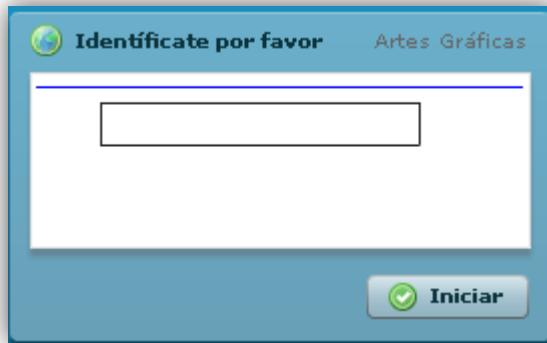


Para continuar ingresaremos nuestros dos textInput dentro de nuestro form y cambias las propiedades Label y ID así como como elemento del Form la opción Require.

Lo que nos resta es agregar los textInput, recuerda que ya tenemos nuestro Form para agregarlos y tener los FormItem

Ya que este dentro vamos a configura la propiedad Horizontal Align a la derecha.





Agregamos los dos text y cambiamos las propiedades de:

- ▶ Label: Usuario
- ▶ Label Contrasenia
- ▶ textInput: txtUser
- ▶ textInput: txtPwd

a los formItem, cambiamos las opciones de Require.

Required: true ▾

Por ahora terminamos con el diseño de la interfaz de nuestra aplicación de firmado al sistema, ahora vamos a validar nuestras cajas de texto, para esto usaremos el componente StringValidator con las siguientes opciones:

```
id="validaString"
source="{txtUser}"
trigger="{cmdIdentificar}"
triggerEvent="click"
property="text"
```

`requiredFieldError="El nombre de usuario es necesario, por favor ;llénalo!"`

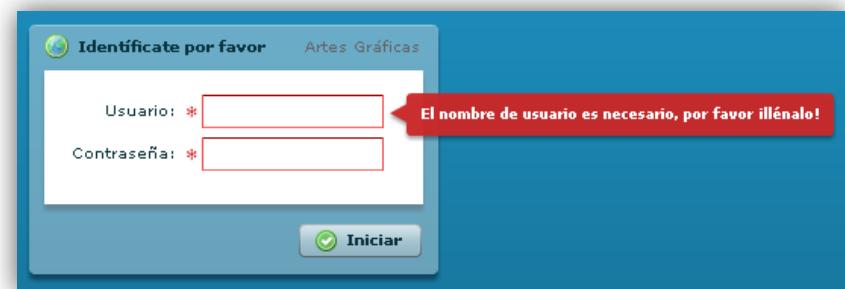
El trigger hace referencia al componente que cuando le pase algo va a disparar la acción del validador, para esto usaremos el botón, y el evento del trigger será "click", así cuando una persona le dé clic al botón de iniciar, tendremos una forma de avisar que algo está sucediendo.

Para cada componente a evaluar, es necesario generar una nueva etiqueta StringValidator, existen diferentes validadores que veremos durante el curso, como lo es el de fecha o el numérico.

El segundo evaluador tendrá características similares, pero en esta ocasión hará referencia al password.

```
id="validaStringPwd"
source="{txtPwd}"
trigger="{cmdIdentificar}"
triggerEvent="click"
property="text"
requiredFieldError="El pasaporte es necesario, por favor ;llénalo!"
```

Ahora con esto podemos probar nuestra aplicación



Aunque ya está evaluado, necesitamos verificar antes de que lanzamos nuestro código, que sería el de verificar. Lo primero que tenemos que hacer es lanzar un método privado al que llamaremos “enviarDatos” que no reciba datos y regrese void.

```
private function enviarDatos():void{  
    |  
}
```

En el cuerpo de esta función evaluaremos los resultados por medio de la creación de una variable de tipo array a la cual asignaremos por medio de la clase estática Validator y el método ValidateAll los elementos a verificar que serán nuestros dos StringValidator.

```
var invalidArray:Array = Validator.validateAll([validaString, validaStringPwd]);
```

Después de que tengamos el resultado lo evaluamos con una sentencia IF en el tamaño de nuestro arreglo, si es igual a 0 entonces todo está en orden de lo contrario existirán algún error.

```
if (invalidArray.length == 0){  
}  
else{  
    Alert.show("Existen errores en el llenado, verifica tus datos","Alerta de Sistema",1);  
}
```

Ahora el código esta completo para lograr la validación completa, lo que nos resta es pedirle hacer algo cuando el IF sea verdadero. En este caso antes de lanzarlo, vamos a generar nuestra conexión con un archivo PHP usando HttpService.

```
<mx:HTTPService id="validaUsuario" result="compruebaHandler(event)"  
    showBusyCursor="true"  
    fault="fallaHandler(event)"  
    method="POST" url="data/comprobar.php" resultFormat="e4x">  
<mx:request>  
    <usuario>  
        {txtUser.text}  
    </usuario>  
    <pwd>  
        {txtPwd.text}  
    </pwd>  
</mx:request>  
</mx:HTTPService>
```

Explicando lo que sucede es un control HTTPService con identificador llamado validaUsuario, el cual tiene un evento en result y uno en fault, el método es POST y el resultado nos lo da en formato E4X.

Aquí mismo usamos una etiqueta request para enviar los parámetros a nuestro archivo PHP. Por ahora conectaremos a nuestra base de datos por medio de un select, para esto requerimos un código de conexión que se lista abajo.

Código conexión.php

```
<?php  
/*  
 * por Carlos Z.O.  
 * para CURSO TALLER MYSQL-PHP-FLEX FEBRERO 2009  
 * Funcion para conectar al server y ver si el  
 * usuario existe en el servidor  
 */  
function conectar() {
```

```

if
(!($link=@mysql_connect("localhost","root","")))
{
    print "No se logro conectar al servicio";
    exit();
}
if (!@mysql_select_db("artescontrol"))
{
    print "Se conecto satisfactoriamente al
servidor pero no a la base de datos, intentarlo más
tarde";
    exit();
}
return $link;
}

?>

```

Lo que esta sucediendo es que estamos abriendo una conexión a nuestro servidor, que en este caso es local y que tiene una base de datos, lo que es importante mencionar es que el usuario aun es root,

Código comprar.php

```

<?php
$user=$_POST['usuario'];/*AQUÍ ESTAMOS RECIBIENDO
RECIBIENDO LA INFORMACIÓN POR MEDIO DE METODO
POST*/
$pwd=$_POST['pwd'];
$pwd=MD5($pwd);/*NUESTRA BASE DE DATOS TIENE
INFORMACION ECRIPADA POR ES ES NECESARIO USAR ESTE
MD5*/
include_once'conexion.php';/*INCLUIMOS LA LIBRERÍA
DE CONEXIÓN Y INSTANCIAMOS POR MEDIO DE LA
FUNCION*/
$netConex=conectar();

```

```

$resultado=mysql_query("SELECT * FROM usuarios
WHERE sesion='$user' and acceso='$pwd',
$netConex);/*GENERAMOS LA SENTENCIA QUE IRA A DAR
AL MANEJADOR Y SER COMPARADA*/
if(mysql_affected_rows($netConex)>0)/*SI EXISTEN
RESULTADOS PARA ESTA CONSULTA CREAMOS EL CODIGO
XML*/
{
    header("Content-type: text/xml");
    print "<?xml version=\"1.0\" encoding=\"utf-
8\"?>\n";
    print "<resultados>\n";
    while (($row =
mysql_fetch_row($resultado)) !=false)
    {
        print "<resultado>";
        print
"<id_usuario>".$row[0]."</id_usuario>\n";
        print "<nombre>".$row[1]."</nombre>\n";
        print "<nivel>".$row[4]."</nivel>\n";
        print "</resultado>\n";
    }
    print "</resultados>";
}

?>

```

Ahora es tiempo de que podamos probar este código, no sin antes haber verificado las cuentas. Como vimos en la sección de la creación del HTTPService, necesitamos las fuciones de result y de fault, que crearemos en seguida, para fault seria la siguiente forma.

```
private function fallaHandler(evento:FaultEvent):void{
    Alert.show(evento.fault.message as String);
}
```

Este método, nos recibe una parámetro del tipo FaultEvent y no nos regresa valores, generamos un Alert.show con la captura del error.

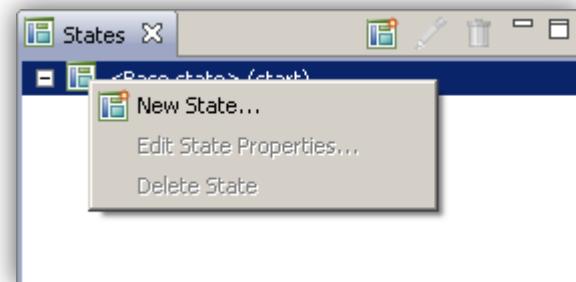
```
private function compruebaHandler(evento:ResultEvent):void{
    var xmlRegreso:XML= new XML(validaUsuario.lastResult);
    if(xmlRegreso.hasComplexContent()){
        Alert.show("Bienvenido al sistema " +
        xmlRegreso..resultado.nombre, "Bienvenido");
        this.currentState="menuAdmin";
        nombreReal=xmlRegreso..resultado.nombre;
        idUsuario=xmlRegreso..resultado.id_usuario;
        nivelUsuario = xmlRegreso..resultado.nivel;

    }else{
        Alert.show(xmlRegreso.toString(), "Error");
    }
}
```

En esta función generaremos un resultado del tipo XML a partir del componente HTTPService y la propiedad LastResult, verificamos si es formato XML y si lo es, Damos la Bienvenida y mandamos a un estado diferente. Previo a esto generaremos variables publicas y Bindable para asignar el nombre del usuario así como su nivel y su ID.

```
[Bindable] public var nombreReal:String;
public var idUsuario:int;
public var nivelUsuario:int;
```

Antes de probar generemos un nuevo estado el cual estará basado en el stado base y eliminaremos el panel child.



Tendremos un estado con un panel igual a estado anterior, lo borramos y tenemos listo el menuAdmin.

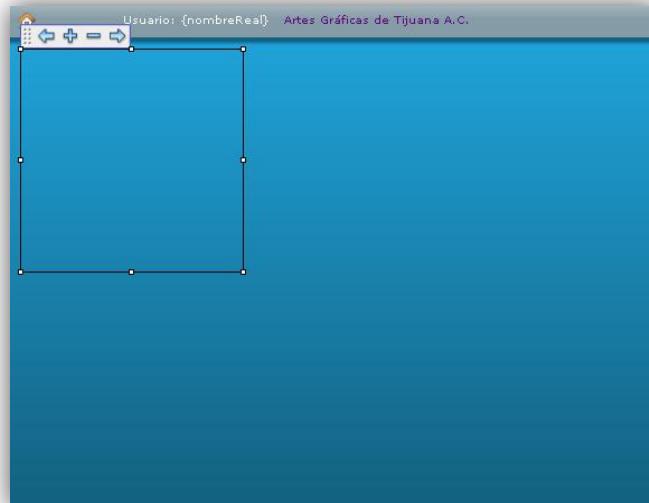
Ahora en el menuAdmin, vamos a agregar un control ApplicationControlBar el cual tendrá las siguientes características.

```
<mx:ApplicationControlBar dock="true" id="menuDock">
```

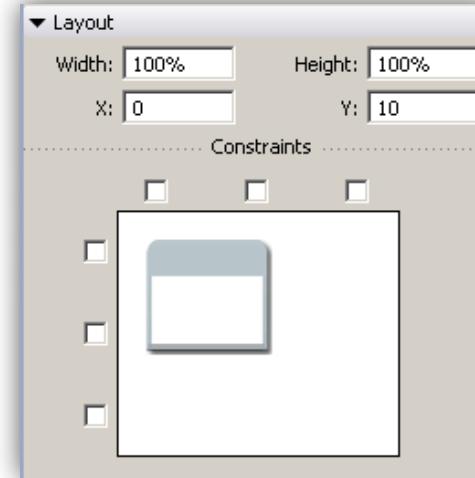
Dentro de este ApplicationControlBar incluimos un control Image, un linkBar y dos Label.

```
<mx:Image source="assets/iconos/house.png"/>
<mx:LinkBar dataProvider="{}"/>
<mx:Label text="Usuario: {nombreReal}" color="#FFFFFF"/>
<mx:Label text="Artes Gráficas de Tijuana A.C." color="#5F0485"/>
```

Recordando que el dataprovider de nuestra LinkBar aun no lo hemos creado. Después de esto generaremos un ViewStack en el area de desarrollo debajo de nuestro ApplicationControlBar.



Los valores de este ViewStack serán de id="paneles" y:

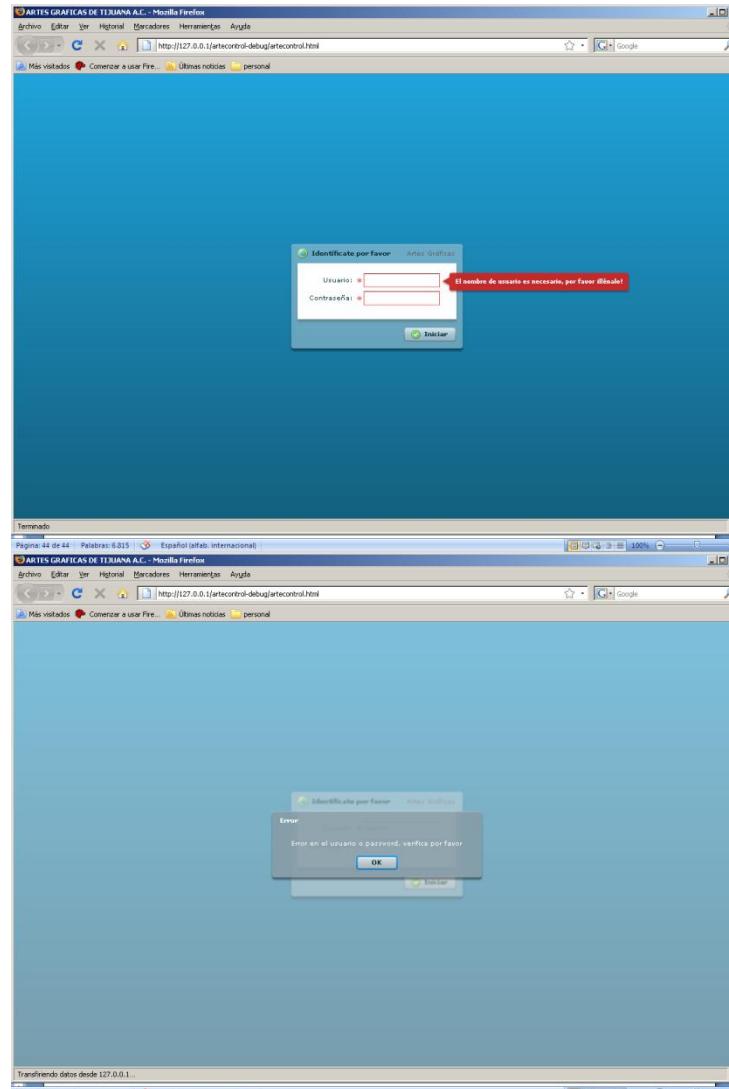
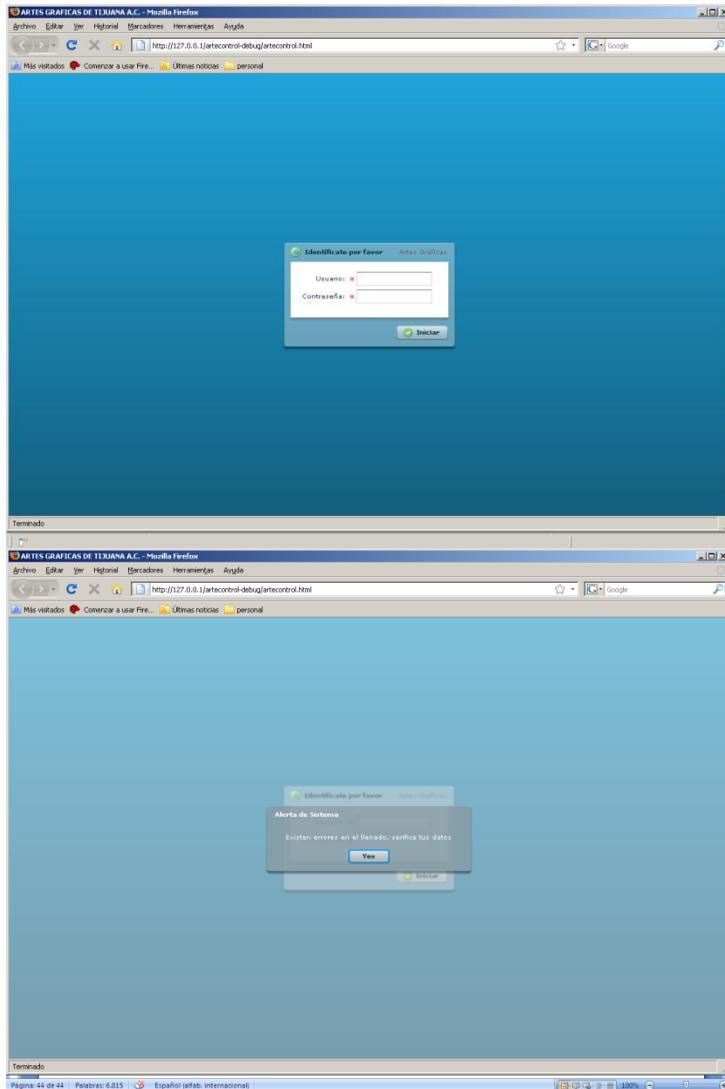


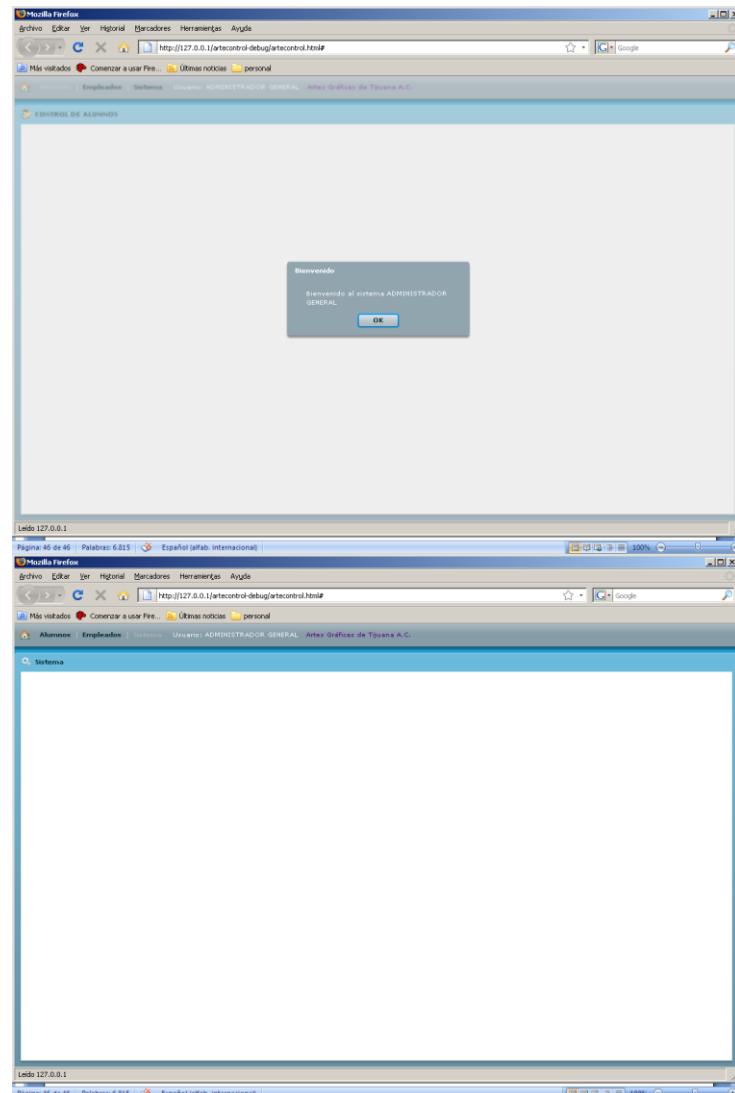
Dentro de nuestro control Paneles ahora generaremos tres paneles

```
<mx:Panel label="Alumnos" width="100%" height="100%" title="CONTROL DE ALUMNOS"
           titleIcon="@Embed(source='assets/iconos/user_edit.png')">
</mx:Panel>
<mx:Panel label="Empleados" width="100%" height="100%" title="Empleados"
           titleIcon="@Embed(source='assets/iconos/user_suit.png')">
</mx:Panel>
<mx:Panel label="Sistema" width="100%" height="100%" title="Sistema"
           titleIcon="@Embed(source='assets/iconos/asignarNL.png')">
</mx:Panel>
```

Con esto terminamos nuestro primer diseño y completamos nuestra aplicación probándola en nuestro server.

► Taller de aplicaciones RIA

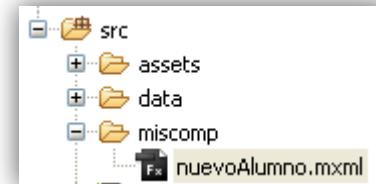


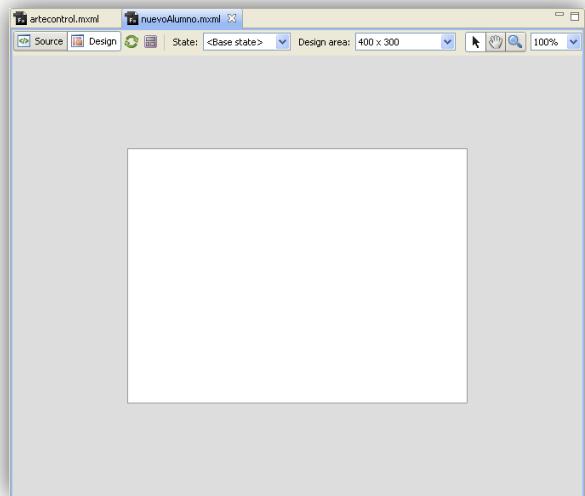


Hasta este momento del taller se ha visto como crear la interfaz de inicio con acceso a una base de datos, validando la información enviada al server.

Creando componente para alta de alumnos

Para realizar esta actividad, vamos a crear una carpeta para todos nuestros componentes, esta se llamará “miscomp”, después creamos un nuevo componente basado en canvas. Eliminamos el tamaño del componente y ponemos como nombre “nuevoAlumno.mxml”





Iniciamos creando todos los componentes necesarios para poder traer información a esta forma, recordando los datos que debemos guardar en nuestro procedimiento.

Name	Type
(+) matricula	char(9)
(+) curp	char(18)
(+) nombre	varchar(25)
(+) paterno	varchar(25)
(+) materno	varchar(25)
(+) sexo	char(20)
(+) f_nacimiento	date
(+) tutor	varchar(80)
(+) telefono	char(13)
(+) celular	char(13)
(+) email	varchar(80)
(+) direccion	varchar(256)
(+) n_interior	int
(+) n_exterior	int
(+) cp	char(5)
(+) municipio_id	int
(+) estado_id	int
(+) fotografia	varchar(25)
(+) actanacimiento	char(2)
(+) comprobante	char(2)
(+) constanciaestudios	char(2)
(+) usuarios_id	int

Para continuar creamos cada uno de los elementos para trabajar con ellos dentro del componente, los cuales terminarán en diseño de esta manera.

Datos principales

Matrícula: Curp:

Nombre: Paterno: Materno:

Sexo: Fecha Nacimiento:

Docimilio & Contacto

Responsable o Tutor:

Teléfono: Celular: Email:

Domicilio: N. Exterior: N. Interior:

CP:

Estado:

Municipio:

Acta de Nacimiento: Comprobante de Domicilio: H. Médico:

Fotografía:

Una vez que terminemos debemos generar el nombre de cada control:

```
maxChars="9" id="txtMatricula"
maxChars="18" id="txtCurp"
id="txtNombre"
id="txtPaterno"
id="txtMaterno"
id="cmbSexo"
id="txtFechaNac"
id="txtResponsable"
id="txtDomicilio"
id="txtExterior"
id="txtInterior"
id="txtTelefono"
id="txtCelular"
id="txtEmail"
maxChars="5" id="txtCP"
id="cmbEstado"
```

```
id="cmbMunicipio"
id="cmbNacimiento"
id="cmbComprobante"
id="cmbMedico"
id="txtFoto"
```

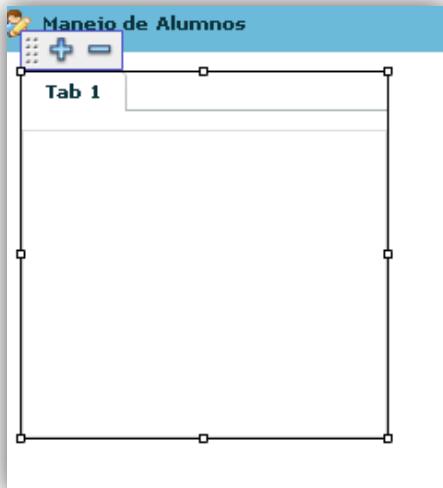
```
<mx:Button label="Guardar" icon="@Embed(source='../assets/iconos/accept.png')" id="cmdGuardar"/>
<mx:Button label="Nuevo" icon="@Embed(source='../assets/iconos/table.png')" id="cmdNuevo"/>
```

Ya que tenemos la interfaz realizaremos los validadores para cada uno de los elementos que se requieren en la interfaz para ser guardados en la base de datos por medio del procedimiento.

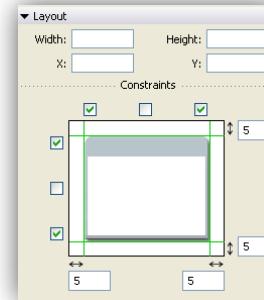
```
<mx:StringValidator id="validaNombre" source="{txtNombre}" trigger="{cmdGuardar}"
triggerEvent="click" property="text" requiredFieldError="El nombre del alumno es requerido"/>
```

Para cada uno de los controles que acabas de agregar. Para probar vamos regresar el archivo principal, donde tenemos el menú.

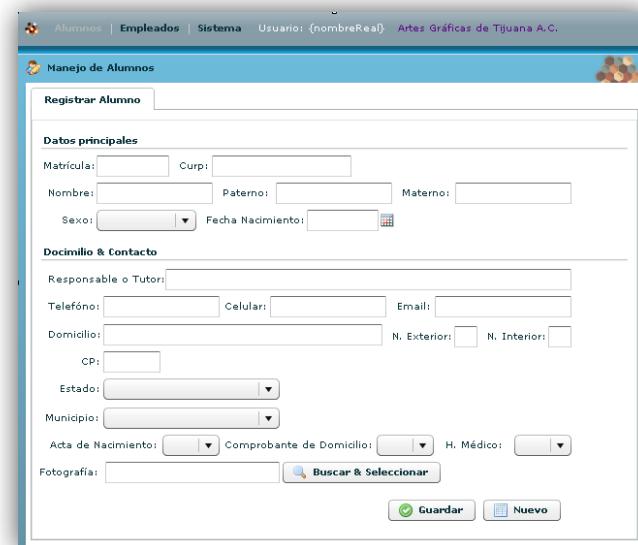
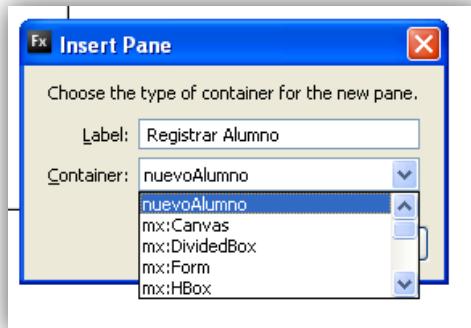
Ahí buscamos el panel de alumnos dentro del ViewStack en el states “menuAdmin”, y el panel donde se manejaran los alumnos, cambiamos la propiedad “Layout” a absolute, en seguida seleccionamos el control “TabNavigator” arrastrándolo sobre el panel.



Seleccionamos el “TabNavigator” y los constraint del lado derecho en la ventana de propiedades.



Damos clic en el icono y agregamos nuestro componente basado en canvas y eliminamos el que está por default.



Ahora que hemos terminado el diseño, vamos a programarlo creando lo siguiente:

ArrayLists

HTTPServices

Validator

Lo primero será el código para llenar los Combos, existen dos tipos, los estáticos y los dinámicos. Para los estáticos vamos a crear variables de tipo ArrayLists.

```
import mx.collections.ArrayList;
[Bindable] private var sexo:ArrayList = new ArrayList([
    {label:"Hombre", data:"H"}, {label:"Mujer", data:"M"}]);
[Bindable] private var siNo:ArrayList = new ArrayList([
    {label:"SI"}, {label:"NO"}]);
```

Label será lo que se muestre en el combo, y data lo que mandaremos al control HTTPService.

Ya teniendo identificado nuestros elementos asignamos en la propiedaddataProvider de cada uno de los elementos asignándolo como Binding.

Esto lo estaremos haciendo dentro de nuestro componente.

Ahora para la parte dinámica, generamos un HTTPServices, para traer la información a nuestra aplicación. Para esto generamos primero el PHP que nos traerá los valores que necesitamos.

```
<mx:HTTPService id="estadosService" showBusyCursor="false"
    result="estadoServiceHandler(event)" fault="fallaHandler(event)" url="data/estados.php"/>
```

Y generamos nuestros métodos para result, el de fault, ya esta creado, usaremos el mismo para el acceso y para los estados.

```
[Bindable] public var estadosAC:ArrayList = new ArrayList();
public function estadoServiceHandler(evento:ResultEvent):void{
    estadosAC = evento.result.estados.estado;
}
```

Y creamos una variable del mismo tipo en nuestro componente para poder pasar esa variable.

```
[Bindable] public var estadosACC:ArrayList;
```

Esta variable no necesita llamarse igual a la que se está generando en la aplicación principal. Y ahora donde incluimos el componente llamaremos a esa propiedad para poder generar el método de llenado.

```
<mx:ComboBox x="74" y="267" width="182" id="cmbEstado"
    dataProvider="{estadosACC}" labelField="estado_n"></mx:ComboBox>
```

Ahora vamos a llenar el combo siguiente cuando tenga seleccionado un estado en el evento “change”.

```
<mx:ComboBox x="74" y="267" width="182" id="cmbEstado" change="cambioDeMunicipio(event)" dataProvider="{estadosACC}" labelField="estado_n"></mx:ComboBox>
```

Generamos el método “cambioDeMunicipio” de tipo private y sin regresar valores el cual tendrá un objeto HttpServices llamado filtroEstado al cual le enviaremos el id_estado del elemento seleccionado por medio de método GET.

```
private function cambioDeMunicipio(evento:Event):void{
    filtroEstado.send({id:evento.target.selectedItem.id_estado});
}
```

```
<mx:HTTPService id="filtroEstado"
    result="filtroEstadoHandler(event)"
    fault="fallaCompAlumno(event)"
    url="data/muniFiltro.php" method="GET"/>
```

Nuestro código en PHP se generaría de esta manera:

```
1 <?php
2 $edo = $_GET['id'];
3 include_once 'conexion.php';
4 $netConex = conectar ();
5 $resultado = mysql_query ("SELECT id_municipio, municipio
6     FROM municipios where estados_id=$edo",$netConex);
7 if (mysql_affected_rows ( $netConex ) > 0)
8 {
9     header ( "Content-type: text/xml" );
10    print "<?xml version=\"1.0\" encoding=\"utf-8\"?>\n";
11    print "<municipios>\n";
12    while ( ($row = mysql_fetch_row ( $resultado )) != false ) {
13        print "<municipio>";
14        print "<id_municipio>" . $row [0] . "</id_municipio>\n";
15        print "<municipio_n>" . $row [1] . "</municipio_n>\n";
16        print "</municipio>\n";
17    }
18    print "</municipios>";
19
20 }
21
22 ?>
```

Cuando tengamos este código lo nombraremos como “muniFiltro.php”, y lo guardaremos en data. Cuando terminemos esta sección vamos a tratar los resultados.

```

private function filtroEstadoHandler(evento:ResultEvent):void{
    var catMunicipios:ArrayCollection = new ArrayCollection();
    try{
        catMunicipios = evento.result.municipios.municipio;
    }catch(error:Error){

    }

    if(catMunicipios.length==0){
        Alert.show("No hay municipios para este estado","Catálogo vacío");
    }else{
        cmbMunicipio.dataProvider = catMunicipios;
    }
}

```

En nuestro éxito de la consulta, trabajaremos con una variable de tipo ArrayCollection para tener los estados.

Generamos un bloque try&catch y asignamos dentro de él, el resultado de la consulta, si todo esta correcto, nuestro ArrayCollection tendrá un tamaño mayor a cero, de lo contrario, estará vacío.

Si no está vacío, se asigna a nuestro cmbMunicipio en la propiedad dataProvider.

Y a nuestro cmbMunicipio, cambiamos el labelFiel por el de municipio_n.



Subiendo archivos al servidor

Ahora nos falta solo generar nuestra búsqueda de archivos y subirlas al server cuando se guarde la información. Para esto utilizaremos transferencia de archivos desde Flex.

Para hacerlo requerimos lo mínimo para mandarlo al server. Solo es necesario tener la información que se requiere en una caja de texto, información para el usuario y un botón para seleccionar.

Para lograr esto usamos la clase FileReference la cual nos permite localizar archivos en los equipos de los clientes con una caja de dialogo dependiendo del sistema operativo. Así mismo despacha los eventos de esta selección.

Necesitamos implementar la clase sobre una variable.

```
var myFileReference:FileReference = new FileReference();
```

Property	Type	Description
creationDate	Date	The creation date of the file on the local computer.
modificationDate	Date	The date the file was last modified on the local computer.
name	String	The name of the file on the local computer.
size	uint	The size, in bytes, of the file on the local computer.
type	String	The file type.

Estas son las propiedades con las que podemos trabajar. Para buscar el archivo dentro de nuestro equipo debemos usar el objeto creado con el método browse().

```
private var miArchivo:FileReference = new FileReference();
private function buscarArchivo():void{
    miArchivo.browse();
}
```

Después asignarle al botón la funcionalidad de este método “buscarArchivo()” el botón a usar es “cmdBuscar”.



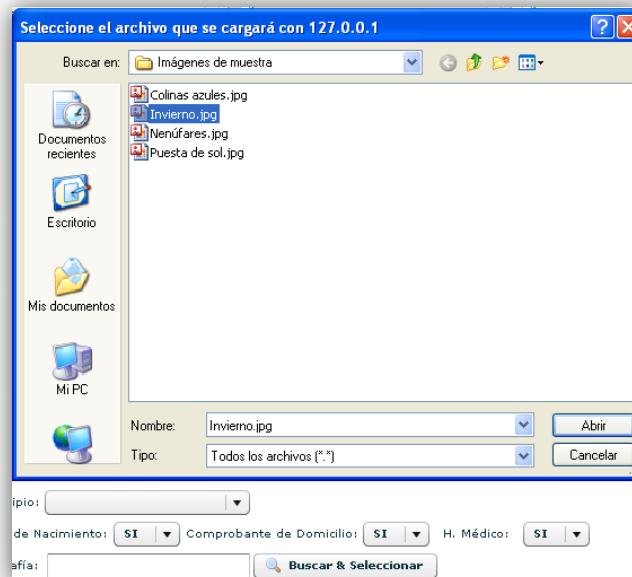
Ahora es tiempo de generar los eventos que estarán regulando la actividad de estas acciones, para esto crearemos un método privado llamado “iniciaEventos” el cual asignaremos al método creationComplete() del canvas que lo contiene.

```
private function iniciaEventos():void{
    miArchivo.addEventListener(Event.SELECT, seleccionarHandler);
}
```

Ahora es tiempo de crear “seleccionarHandler()”, que sucede cuando tomamos el archivo del cuadro de dialogo.

```
private function seleccionarHandler(evento:Event):void{
    Alert.show("Seleccionaste el archivo "+miArchivo.name,"Información de Archivo");
    txtFoto.text = miArchivo.name;
}
```

Lo que sucede aquí es cuando se selecciona el archivo, se manda un mensaje al usuario que acaba de seleccionar archivo y tiene un nombre, el mismo que colocaremos en la caja de texto “txtFoto”.



El siguiente paso es generar el método que suba la imagen al servidor, esta será el botón de guardar.

```
private function subirArchivoAlServer(nombre:String):void{
    var peticion:URLRequest = new URLRequest("data/subirFoto.php");
    peticion.method = URLRequestMethod.POST;
    var variables:URLVariables = new URLVariables();
    variables.nameFile = nombre;
    peticion.data = variables;
    miArchivo.upload(peticion);
}
```

Se crea una variable “peticion” de tipo URLRequest y se le asigna al constructor la dirección de donde se encuentra el script para guardar en el servidor, recordando de que PHP hará el trabajo del lado del server nuestro script será “subirFoto.php”, Este script estará formado por la siguiente información.

```
$nombre=$_POST['nameFile'];
foreach ($_FILES as $fieldName => $file) {
    echo move_uploaded_file($file['tmp_name'], "../assets/pic/" . $nombre);
}
```

Crearemos una carpeta llamada “pic” dentro de la carpeta de “assets” y le damos permisos de lectura y escritura.

Regresando al método con el objeto “peticion” usamos el método POST y creamos un objeto de tipo URLVariables, para poder enviar información adicional al server.

Le asignamos los valores y usamos el método “upload” para enviar a información al servidor.

En el método de los eventos creamos uno nuevo llamado “completaSubidaHandler”.

```
private function inicioEventos():void{
    miArchivo.addEventListener(Event.SELECT, seleccionarHandler);
    miArchivo.addEventListener(Event.COMPLETE, completaSubidaHandler);
}
```

Este es el método que nos dirá que todo está correcto.

```
private function completaSubidaHandler(evento:Event):void{
    Alert.show("El archivo ha sido subido con éxito", "Felicitaciones");
}
```

Terminado esta parte del manual, requerimos ajustar unos pequeños detalles, este es la fecha, recordando el formato que MySQL recibe “YYYY-mm-dd”, vamos a configurar el control de la fecha de nacimiento.

```
formatString="DD/MM/YYYY"
dayNames="['D', 'L', 'M', 'M', 'J', 'V', 'S']"
maxYear="2010"
yearNavigationEnabled="true"
monthNames="['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio', 'Julio', 'Agosto', 'Septiembre', 'Octubre', 'Noviembre', 'Diciembre']"
```



Ahora solo queda generar el HTTPService para guardar la información.

```
<mx:HTTPService id="guardarAlumno"
    result="guardarAlumnoHandler(event)"
    fault="fallaCompAlumno(event)"
    url="data/saveAlumno.php" method="POST">
```

Este archivo “saveAlumno.php” deberá tener el acceso a la información recordando que se usará un procedimiento almacenado

```
<?php
include_once 'conexion.php';
$netConex = conectar();
$user = $_POST ['id_usuario'];
$nombre = $_POST ['nombre'];
$paterno =$_POST ['paterno'];
$materno = $_POST ['materno'];
$sexo = $_POST ['sexo'];
$matricula =$_POST ['matricula'];
$curp = $_POST ['curp'];
$fecha_nac =$_POST ['fechanac'];
$responsable =$_POST ['responsable'];
$domicilio = $_POST ['domicilio'];
$telefono = $_POST ['telefono'];
$celular = $_POST ['celular'];
$email = $_POST ['email'];
$exterior =$_POST ['exterior'];
$interior =$_POST ['interior'];
$cp = $_POST ['cp'];
$estado =$_POST ['estado'];
$municipio = $_POST ['municipio'];
$acta = $_POST ['acta'];
$comprobante =$_POST ['comprobante'];
$constancia = $_POST ['constancia'];
$foto =$_POST['foto'];
```

```
$cadena = "CALL sp_guardarAlumno
('$_matricula','$_curp','$_nombre','$_paterno','$_materno','$_exo',
'$_fecha_nac','$_responsable','$_telefono',
'$_celular','$_email','$_domicilio',$_interior,$_exterior,'$_cp',
$_municipio,$estado,'$_foto','$_acta','$_comprobante',
'$_constancia',$_user);";

echo mysql_query($cadena,$netConex) or die("No se
guarda");

?>
```

Este procedimiento nos permitirá guardar nuestra información en el server. Para guardar el alumno, una de las maneras es saber si ya se subió el archivo con el curp del alumno, si lo hizo con éxito, entonces guardar la información, para esto en el método “completaSubidaHandler” vamos a agregar el siguiente código.

```
guardarAlumno.send({
    matricula:txtMatricula.text,
    curp:txtCurp.text,
    nombre:txtNombre.text,
    paterno:txtPaterno.text,
    materno:txtMaterno.text,
    sexo:cmbSexo.selectedItem.data,
    fechanac:txtFechaNac.text,
    responsable:txtResponsable.text,
    telefono:txtTelefono.text,
    celular:txtCelular.text,
    email:txtEmail.text,
    domicilio:txtDomicilio.text,
    interior:txtInterior.text,
    exterior:txtExterior.text,
    cp:txtCP.text,
    estado:cmbEstado.selectedItem.id_estado,
    municipio:cmbMunicipio.selectedItem.id_municipio,
    acta:cmbNacimiento.selectedLabel,
    comprobante:cmbComprobante.selectedLabel,
    constancia:cmbConstancia.selectedLabel,
```

```
foto:txtCurp.text + ".jpg",
id_usuario:idUsuarioC
});
```

Al lanzar esta información al server, lo procesa y regresa la información en “guardarAlumnoHandler”.

En este método verificamos si el regreso “evento.result” es igual al texto “No se guarda” y haciendo uso de un if podemos mandar el mensaje correcto.

```
private function guardarAlumnoHandler(evento:ResultEvent):void{
    if (evento.result == "No se guarda"){
        Alert.show("El alumno NO se guardo en el sistema ","Error");
    }else{
        Alert.show("El alumno se guardo en el sistema ","Felicitaciones");
    }
}
```

Necesitamos validar esto lo haremos en el método “subirArchivoAlServer” antes de que hagamos el .upload(), el código seria por medio de un Validator.validateAll

```
private function subirArchivoAlServer():void{
    var peticion:URLRequest = new URLRequest("data/subirFoto.php");
    peticion.method = URLRequestMethod.POST;
    var variables:URLVariables = new URLVariables();
    variables.nameFile = txtCurp.text + ".jpg";
    peticion.data = variables;
    var arreglo:Array = Validator.validateAll([
        validaMatricula,
        validaCP,
        validaCurp,
        validaDomicilio,
        validaFechaNac,
        validaFoto,
        validaMaterno,
        validaNombre,
        validaPaterno,
        validaResponsable
    ]);
    if (arreglo.length>0){
        Alert.show("Existen errores en el llenado de los datos","Verifica datos");
    }else{
        miArchivo.upload(peticion);
    }
}
```

Y por ultimo vamos a limpiar las cajas y reiniciar las cajas y combos.

```

private Function limpiarCajas():void{
    txtMatricula.text= "";
    txtCurp.text= "";
    txtNombre.text= "";
    txtPaterno.text= "";
    txtMaterno.text= "";
    cmbSexo.selectedIndex=0
    txtFechaNac.text= "";
    txtResponsable.text= "";
    txtTelefono.text= "";
    txtCelular.text= "";
    txtEmail.text= "";
    txtDomicilio.text= "";
    txtInterior.text= "";
    txtExterior.text= "";
    txtCP.text= "";
    cmbEstado.selectedIndex=0
    cmbMunicipio.dataProvider="";
    cmbNacimiento.selectedIndex=0;
    cmbComprobante.selectedIndex=0
    cmbConstancia.selectedIndex=0
    txtFoto.text = "";
    txtMatricula.setFocus();
}

```

Este método lo lanzaremos en el clic del botón Nuevo y cuando mande el mensaje de éxito al guardar.

Componente para inscribir a los alumnos registrados

En esta sección generaremos el código para poder inscribir a un alumno, así como asignarle las materias que le corresponden. El trabajo final de la interfaz será:

La funcionalidad de este componente estará afectando directamente a los procedimientos “sp_guardarInscripcion” y “sp_guardarMovimiento”. Estos procedimientos están generados en nuestra base de datos, los controles nuevos que se implementaran en esta forma, será un dataGrid.

El primer paso es diseñar la interfaz. Los controles a usar son los siguientes:

```

<mx:HRule y="29" left="10" right="10"/>
<mx:Label x="10" y="13" text="Datos principales"
fontWeight="bold"/>
<mx:Label x="10" y="39" text="Matrícula:" textAlign="right"/>

```

```

<mx:TextInput x="67" y="37" maxChars="9" width="85"
id="txtMatricula"/>
<mx:Label x="196" y="123" text="Cantidad:" textAlign="right"/>
<mx:TextInput x="256" y="121" maxChars="9" width="85"
id="txtCantidad" restrict="0-9."/>
<mx:Label x="192" y="37" text="Curnp:" textAlign="right"/>
<mx:TextInput x="228" y="35" width="144" id="txtCurnp"
enabled="false"/>
<mx:Label x="15" y="67" text="Nombre:" textAlign="right"/>
<mx:TextInput x="67" y="65" width="120" id="txtNombre"
enabled="false"/>
<mx:Label x="195" y="67" text="Paterno:" textAlign="right"/>
<mx:TextInput x="252" y="65" width="120" id="txtPaterno"
enabled="false"/>
<mx:Label x="380" y="67" text="Materno:" textAlign="right"/>
<mx:TextInput x="437" y="65" width="120" id="txtMaterno"
enabled="false"/>
<mx:HRule y="111" left="10" right="10"/>
<mx:Label x="10" y="95" text="Inscripción" fontWeight="bold"/>
<mx:Label x="29" y="151" text="Curso:" textAlign="right"/>
<mx:ComboBox x="74" y="149" width="235"
id="cmbCurso"></mx:ComboBox>
<mx:Label x="10" y="123" text="Semestre:" textAlign="right"/>
<mx:ComboBox x="74" y="121" width="114" id="cmbSemestre"
></mx:ComboBox>

<mx:Button label="Guardar"
icon="@Embed(source='../assets/iconos/accept.png')"
id="cmdGuardar" x="379" y="149"/>

<mx:Button icon="@Embed(source='../assets/iconos/buscar.png')"
id="cmdBuscarMatricula"
enabled="{txtMatricula.text!=''}" x="160" y="36" width="24"
click=""/>

<mx:Button label="Nuevo"
icon="@Embed(source='../assets/iconos/table.png')" id="cmdNuevo"
click="" x="477" y="149"/>

<mx:DataGrid x="10" y="200" width="547" height="200"
id="grdCargaMaterias">
<mx:columns>
<mx:DataGridColumn width="100" headerText="Clave"
dataField="id_materia"/>

```

```

<mx:DataGridColumn width="447" headerText="Materia"
dataField="descripcion"/>
</mx:columns>
</mx:DataGrid>

<mx:HRule y="193" left="10" right="10"/>
<mx:Label x="10" y="177" text="Carga de materias"
fontWeight="bold"/>
<mx:Button
icon="@Embed(source='../assets/iconos/refreshGrid.png')"
id="cmdCargarMaterias" x="317" y="149" width="24"
click=""/>

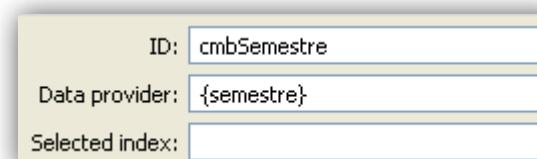
```

Una vez creada la interfaz vamos a programar cada uno de los elementos, primero vamos a generar el ArrayCollection para el listado de semestres.

```

[Bindable] private var semestre:ArrayCollection = new ArrayCollection(
[{"label": "Primero", "data": "1"}, {"label": "Segundo", "data": "2"}, {"label": "Tercero", "data": "3"}, {"label": "Cuarto", "data": "4"}, {"label": "Quinto", "data": "5"}, {"label": "Sexto", "data": "6"}]);

```



Y se lo asignamos al combo con el nombre respectivo.

El siguiente paso es llenar el listado de cursos disponibles en la institución para esto usaremos un PHP y traeremos por medio de un HTTPServices la información a mostrar, lo primero será el PHP.

```
<?php
include_once 'conexion.php'; /*INCLUIMOS LA LIBRERÍA DE CONEXIÓN
Y INSTANCIAMOS POR MEDIO DE LA FUNCION*/
$netConex = conectar ();
$resultado = mysql_query ("SELECT id_curso, nombre_curso FROM
cursos");
if (mysql_affected_rows ( $netConex ) > 0)
{
    header ( "Content-type: text/xml" );
    print "<?xml version=\"1.0\" encoding=\"utf-8\"?>\n";
    print "<cursos>\n";
    while ( ($row = mysql_fetch_row ( $resultado )) != false
) {
        print "<curso>";
        print "<id_curso>" . $row [0] . "</id_curso>\n";
        print "<nombre_curso>" . $row [1] .
"</nombre_curso>\n";
        print "</curso>\n";
    }
    print "</cursos>";
}
?>
```

Para probar el resultado podemos verlo en la dirección donde hacemos la publicación /data/cursos.php.

```
- <cursos>
- <curso>
<id_curso>1</id_curso>
<nombre_curso>DANZA</nombre_curso>
</curso>
- <curso>
<id_curso>2</id_curso>
<nombre_curso>MUSICA</nombre_curso>
</curso>
- <curso>
<id_curso>3</id_curso>
<nombre_curso>PINTURA</nombre_curso>
</curso>
- <curso>
<id_curso>4</id_curso>
<nombre_curso>ESCULTURA</nombre_curso>
</curso>
</cursos>
```

Generamos nuestro HTTPService en el cual generamos dos métodos de regreso, estos serán listadoCursoHandler para el result y fallaCompInscripcion para todos los fault.

```
<mx:HTTPService id="listadoCurso"
result="listadoCursoHandler(event)"
fault="fallaCompInscripcion(event)"
url="data/cursos.php"/>
```

```
private function fallaCompInscripcion(evento:FaultEvent):void{
    Alert.show(evento.fault.faultString, "Error")
}
private function listadoCursoHandler(evento:ResultEvent):void{
    cmbCurso.dataProvider = evento.result.cursos.curso;
}
```

En la función `listadoCursoHandler` regresamos el valor directamente al `comboBox` con la propiedad `dataProvider`. Probamos.



Ahora es tiempo de generar el código para traer la validación del alumno por medio de su matrícula, esta será enviada al server para ver si se encuentra registrado, de ser así traerá los datos como el nombre completo y su curp, de lo contrario, nos mandara un error de búsqueda.

Primero generamos el código PHP que nos filtre la información que requerimos:

```
<?php
$matricula = $_POST['m'];
include_once 'conexion.php'; /*INCLUIMOS LA LIBRERÍA DE CONEXIÓN
Y INSTANCIAMOS POR MEDIO DE LA FUNCIÓN*/
$netConex = conectar ();
$resultado = mysql_query ("select curp,nombre,paterno,materno
from alumnos where matricula='".$matricula"'");
if (mysql_affected_rows ( $netConex ) > 0)
{
    header ( "Content-type: text/xml" );
    print "<?xml version=\"1.0\" encoding=\"utf-8\"?>\n";
    print "<alumnos>\n";
    while ( ($row = mysql_fetch_row ( $resultado )) != false
) {
        print "<alumno>";
        print "<curp>" . $row [0] . "</curp>\n";
        print "<nombre>" . $row [1] . "</nombre>\n";
        print "<paterno>" . $row [2] . "</paterno>\n";
        print "<materno>" . $row [3] . "</materno>\n";
        print "</alumno>\n";
    }
    print "</alumnos>";
}
else print "no";
?>
```

```
print "<nombre>" . $row [1] . "</nombre>\n";
print "<paterno>" . $row [2] . "</paterno>\n";
print "<materno>" . $row [3] . "</materno>\n";
print "</alumno>\n";
}
print "</alumnos>";
```

Ahora es tiempo de generar nuestro `HTTPServices` para esta sección:

```
<mx:HTTPService id="buscarAlumno"
result="buscarAlumnoHandler(event)"
fault="fallaCompInscripcion(event)"
url="data/buscarAlumno.php"
method="POST"/>
```

Generamos nuestro método de result llamado “`buscarAlumnoHandler`”:

```
private function buscarAlumnoHandler(evento:ResultEvent):void{
    try{
        if(evento.result=="no"){
            Alert.show("No se encuentra la matrícula "+txtMatricula.text+" "+
            "en el sistema","Alumno no encontrado");
        }else{
            txtNombre.text = evento.result.alumnos.alumno.nombre;
            txtPaterno.text = evento.result.alumnos.alumno.paterno;
            txtMaterno.text = evento.result.alumnos.alumno.materno;
            txtCurp.text = evento.result.alumnos.alumno.curp;
        }
    }catch(error:Error){}
}
```

Con este estamos regresando la información nuestro componente a cada caja de texto y de lo contrario el mensaje de que la matrícula buscada no se encuentra en el servidor. Probamos



Ahora nos queda cargar la consulta de datos por medio del semestre y el curso con el clic sobre el botón .

Hacemos el PHP para lanzar nuestro filtro:

```
<?php
$curso = $_POST['c'];
$semestre = $_POST['s'];
include_once 'conexion.php'; /*INCLUIMOS LA LIBRERÍA DE CONEXIÓN
Y INSTANCIAMOS POR MEDIO DE LA FUNCION*/
$netConex = conectar ();
$resultado = mysql_query ("SELECT DISTINCT
    materias.semestre,
    materias.descripcion,
    materias.id_materia,
    cursos.id_curso
FROM
    materias
INNER JOIN cursos ON (materias.pertenece_curso=cursos.id_curso)
WHERE cursos.id_curso=$curso and
materias.semestre='$semestre'");
if (mysql_affected_rows ( $netConex ) > 0)
{
    header ( "Content-type: text/xml" );
    print "<?xml version=\"1.0\" encoding=\"utf-8\"?>\n";
    print "<materias>\n";
    while ( ($row = mysql_fetch_row ( $resultado )) != false
) {
        print "<materia>";
        print "<semestre>" . $row [0] . "</semestre>\n";
        print "<descripcion>" . $row [1] .
"</descripcion>\n";
        print "<id_materia>" . $row [2] .
"</id_materia>\n";
        print "<id_curso>" . $row [3] . "</id_curso>\n";
        print "</materia>\n";
    }
    print "</materias>";
}
?>
```

Ahora es tiempo del **HTTPService** el cual tendrá nuestro result dirigido al método "materiasHandler"

```
<mx:HTTPService id="materias"
    result="materiasHandler(event)"
    fault="fallaCompInscripcion(event)"
    url="data/materiasXsemestre.php"
    method="POST"/>
```

```
private function materiasHandler(evento:ResultEvent):void{
    grdCargaMaterias.dataProvider = evento.result.materias.materia;
}
```

y este resultado estará regresando al datagrid. Ahora solo es cuestión de lanzar el **HTTPService**:

```
<mx:Button icon="@Embed(source='../assets/iconos/refreshGrid.png')"
    id="cmdCargarMaterias" x="317" y="149" width="24"
    click="materias.send({s:cmbSemestre.selectedItem.data,c: cmbCurso.selectedItem.id_curso})"/>
```

The screenshot shows a user interface for managing subjects. At the top, there are dropdown menus for 'Semestre' (set to 'Primero') and 'Curso' (set to 'DANZA'), along with buttons for 'Guardar' (Save) and 'Nuevo' (New). Below these is a section titled 'Carga de materias' containing a table with columns 'Clave' and 'Materia'. The table data is as follows:

Clave	Materia
1	COMPRESION DEL ARTE
3	ACONDICIONAMIENTO
5	TECNICA MODERNA 1
6	TECNICA CLASICA 1
7	TECNICA MODERNA EJECUTANTE 1

```
private function limpiarCajas():void{
    txtMatricula.text= "";
    txtCurnp.text= "";
    txtNombre.text= "";
    txtPaterno.text= "";
    txtMaterno.text= "";
    txtCantidad.text= "";
    txtMatricula.setFocus();
}
```

Y lo que nos restaría será enviar la información y validar esta. Para validar usamos:

```
<mx:StringValidator id="validoCantidad" source="{txtCantidad}" trigger="{cmdGuardar}"
    triggerEvent="click" property="text" requiredFieldError="Introduce la cantidad de inscripción"/>
<mx:StringValidator id="validoMatricula" source="{txMatricula}" trigger="{cmdGuardar}"
    triggerEvent="click" property="text" requiredFieldError="Por favor ingresa la matrícula"/>
```

Y generamos un método para guardar esta información llamado "guardarInscripcionBtn":

Limpieando cajas de texto:

```

private function guardarInscripcionBtn():void{
    var arreglo:Array = Validator.validateAll([
        validaMatricula,
        validaCantidad
    ]);
    if (arreglo.length>0){
        Alert.show("Existen errores en el llenado de los datos","Verifica datos");
    }else{
        guardarInscripcion.send({
            semestre:cmbSemestre.selectedItem.data,
            curso:cmbCurso.selectedItem.id_curso,
            matricula:txtMatricula.text,
            id_usuario:idUsuarioC,
            cantidad:txtCantidad.text
        })
    }
}

```

```

<?php
include_once 'conexion.php';
$netConex = conectar();
$user = $_POST ['id_usuario'];
$matricula =$_POST ['matricula'];
$semestre = $_POST ['semestre'];
$curso = $_POST['curso'];
$cantidad = $_POST['cantidad'];
$cadena1 = "CALL sp_guardarInscripcion
('$semestre',$curso,'$matricula','$user');";
$cadena2= "CALL sp_guardarMovimiento
($cantidad,$user,'$matricula');";
mysql_query($cadena1,$netConex) or die("No se guardo");
mysql_query($cadena2,$netConex) or die("No se guardo");

?>

```

Para revisar verificamos que exista un alumno en la base de datos y con su matrícula la buscamos en nuestra aplicación esta nos traerá la información general y podemos ahora si, inscribir a este alumno a un semestre y curso. Así como cargar las materias que en este caso serán de danza. Semestre I.

2009DZ001	ZAOC810304HDFRRRC	CARLOS	ZARAGOZA	ORTIZ	H	04/03/1981	24/02/201R
2009DZ002	GAGA040429MHGRRN	ANAH	GARCIA	GARCIA	M	04/03/1981	24/02/201R
2009DZ003	GAGA040429MHGRRN	ANA	ASD	DSA	M	18/02/2009	24/02/201A
2009DZ005	ZAOC812025JHGDD8	dsa	asd	asd	H	18/02/2009	24/02/201C

Verificamos que nuestra base de datos los movimientos

	id_movimiento	tipomov_id	cantidad	f_mov	usuario_id	alumno_id
Click here to define a filter						
	1	4	450	25/02/2009	1	2009DZ001
	2	4	450	25/02/2009	1	2009DZ002

Antes de que ejecutemos nuestro guardado. Ahora en la tabla de saldo:

alumno_id	saldos	usuario_id
Click here to define a filter		
2009DZ001	450	1
2009DZ002	450	1
2009DZ003	0	1
2009DZ005	0	1

Ya que el vamos a guardar será el de matricula 2009DZ003.

id_alumno_calif	grupo_id	matricula	cal_parcial	cal_global	cal_final	usuario_id
Click here to define a filter						
1	1	2009DZ001	0	0	0	1
2	2	2009DZ001	0	0	0	1
3	3	2009DZ001	0	0	0	1
4	4	2009DZ001	0	0	0	1
5	5	2009DZ001	0	0	0	1
8	1	2009DZ002	0	0	0	1
9	2	2009DZ002	0	0	0	1
10	3	2009DZ002	0	0	0	1
11	4	2009DZ002	0	0	0	1
12	5	2009DZ002	0	0	0	1

En tabla de calificaciones de alumno “alumno_calif” que no tenga el registro de la matrícula 2009DZ003.

Clave	Materia
1	COMPRESIÓN DEL ARTE
3	ACONDICIONAMIENTO
5	TECNICA MODERNA 1
6	TECNICA CLASICA 1
7	TECNICA MODERNA EJECUTANTE 1

Una vez lanzada la orden nuestra base de datos se debe ver afectada con lo siguientes elementos.

	1	1	2009DZ001	0	0	0
	2	2	2009DZ001	0	0	0
	3	3	2009DZ001	0	0	0
	4	4	2009DZ001	0	0	0
	5	5	2009DZ001	0	0	0
►	8	1	2009DZ002	0	0	0
	9	2	2009DZ002	0	0	0
	10	3	2009DZ002	0	0	0
	11	4	2009DZ002	0	0	0
	12	5	2009DZ002	0	0	0
	15	1	2009DZ003	0	0	0
	16	2	2009DZ003	0	0	0
	17	3	2009DZ003	0	0	0
	18	4	2009DZ003	0	0	0
	19	5	2009DZ003	0	0	0

	►	2009DZ001	450
		2009DZ002	450
		2009DZ003	480
		2009DZ005	0
			1
			1
			1

De esta manera sabemos que se ha guardado satisfactoriamente el alumno dentro de nuestra base de datos con todos los requisitos para ingreso dentro de esta.

Creando catálogos para nuestra base de datos

Estos catálogos son los que darán movimiento a nuestra base de datos, dentro de estas tablas están las siguientes:

- Cursos
- Estados
- Municipios
- Tipoempleado
- tipoMov
- usuarios

	1	4	450	25/02/2009	1	2009DZ001
	2	4	450	25/02/2009	1	2009DZ002
►	3	4	480	25/02/2009	1	2009DZ003

Son seis catálogos que se crearan dentro de la pestaña de sistema. Aquí se aplicaran los conceptos vistos en los días anteriores.

Se requiere de `HTTPService`, del manejo del `result`, creación de `ArrayLists`. Y del método `SEND` con la información.
Iniciamos con CURSOS:



```

<mx:HRule y="29" left="10" right="10"/>
    <mx:Label x="10" y="13" text="Agregar nuevos cursos o
carreras" fontWeight="bold"/>
        <mx:Label x="10" y="41" text="Nombre:"
textAlign="right"/>
            <mx:TextInput y="39" id="txtNombre" left="62"
right="108"/>
                <mx:DataGrid right="10" left="10" top="67" bottom="10">
                    <mx:columns>
                        <mx:DataGridColumn headerText="Clave del
Curso" dataField="col1"/>
                            <mx:DataGridColumn headerText="Nombre del
Curso" dataField="col2"/>
                    </mx:columns>
                </mx:DataGrid>
                <mx:Button id="btnGuardar" y="39" label="Guardar"
icon="@Embed(source='../assets/iconos/accept.png')" right="10"/>

```

Con este código generamos nuestro primer componente, recuerden crearlo en la carpeta de “miscomp”, se crea basado en Canvas sin tamaño definido.

Creamos el PHP para el insert sobre la base de datos usando el procedimiento “sp_guardarCurso”.

```

<?php
include_once 'conexion.php';
$netConex = conectar();
$user = $_POST ['id_usuario'];
$nombre = $_POST ['nombre'];
$cadena = "CALL sp_guardarCurso ('$nombre',$user);";
echo mysql_query($cadena,$netConex) or die("No se guardo");
?>

```

Ahora es tiempo de generar el HTTPService con las opciones indicadas:

```

<mx:HTTPService id="guardarCurso"
    result="guardarCursoHandler(event)"
    fault="fallaCompCursos(event)"
    method="POST" url="data/saveCurso.php"/>

```

Generamos los métodos result y fault:

```

private function guardarCursoHandler(evento:ResultEvent):void{
}

private function fallaCompCursos(evento:FaultEvent):void{
    Alert.show(evento.fault.faultString, "Error");
}

```

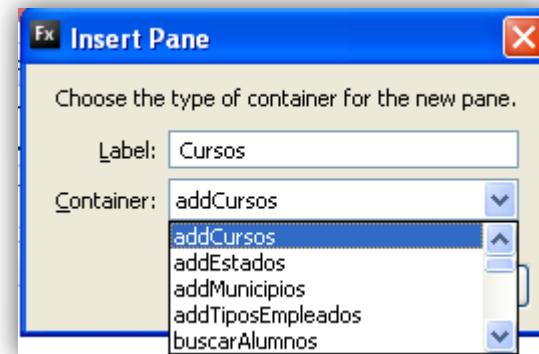
Más adelante daremos funcionalidad al método “guardarCursoHandler”, por ahora crearemos los StringValidator

```
<mx:StringValidator
    id="validaNombreCurso"
    source="{txtNombre}"
    trigger="{btnGuardar}"
    triggerEvent="click"
    property="text"/>
```

Ahora creamos el método para guardar este proceso, lo llamaremos "guardarElCurso"

```
private function guardarElCurso():void{
    var verifica:Array = Validator.validateAll([validaNombreCurso]);
    if (verifica.length>0){
        Alert.show("Existen errores de llenado de datos, verifica","Error");
    }else{
        guardarCurso.send({nombre:txtNombre.text,id_usuario:idUsuarioC});
    }
}
```

Ahora es cuestión de hacer un debug sobre el regreso es decir en el método "guardarCursoHandler". Es tiempo de agregarlo en nuestra aplicación principal.



Ahora pasamos el id del usuario para poder recibirla en nuestra catalogo. Es tiempo de probar la aplicación y ver si regresa algún dato en el result.

```
private function guardarCursoHandler(evento:ResultEvent):void{
    if(evento.result==1){
        Alert.show("Se guardo correctamente","Exito");
    }else{
        Alert.show("Error en el guardado","Exito");
    }
}
```

Es tiempo de generar el listado de cursos y hacer un refresh sobre el send de este listado así como limpiar la caja de texto txtNombre para estar lista a ser agregado otro elemento.

Actividad: REALIZAR LOS OTROS 5 CATALOGOS CON LA MISMA FUNCIONALIDAD.

Búsqueda de alumnos desde un dataGrid

Nuestra siguiente aplicación será una llamada “buscarAlumnos” basada en canvas sin tamaño definido. En esta ocasión trabajaremos un segundo estado dentro de un componente.

Primer paso será crear un filtro de alumnos.

```
<?php
$dato = $_POST['d'];
include_once 'conexion.php';
$netConex = conectar ();
$resultado = mysql_query ("select
matricula,curp,nombre,paterno,materno
from alumnos where nombre like '$dato%' or paterno like '$dato%
or materno like '$dato%' and matricula like'$dato'");
if (mysql_affected_rows ( $netConex ) > 0)
{
    header ( "Content-type: text/xml" );
    print "<?xml version=\"1.0\" encoding=\"utf-8\"?>\n";
    print "<alumnos>\n";
    while ( ($row = mysql_fetch_row ( $resultado )) != false
) {
        print "<alumno>";
        print "<matricula>" . $row [0] .
"</matricula>\n";
        print "<curp>" . $row [0] . "</curp>\n";
        print "<nombre>" . $row [1] . "</nombre>\n";
        print "<paterno>" . $row [2] . "</paterno>\n";
        print "<materno>" . $row [3] . "</materno>\n";
        print "</alumno>\n";
    }
    print "</alumnos>";

} else print "0";
}
```

?>
Una vez terminado el filtro, creamos nuestra interfaz, la cual estará en primer plano
una sola sección de texto un botón.

Ahora crear el estado al que llamaremos resultados

El código para crear esta interfaz es la siguiente:

```
<mx:states>
    <mx:State name="resultados">
        <mx:SetProperty target="{label1}" name="text" value="Materias del alumno"/>
        <mx:RemoveChild target="{txtDato}" />
        <mx:RemoveChild target="{btnBuscar}" />
        <mx:RemoveChild target="{dtgResultadosAlumno}" />
        <mx:AddChild position="lastChild">
            <mx:Label x="10" y="39" text="Matrícula:" textAlign="right" />
        </mx:AddChild>
```

```
<mx:AddChild position="lastChild">
    <mx:TextInput x="67" y="37"
maxChars="9" width="117" id="txtMatricula" enabled="true"
editable="false"/>
    </mx:AddChild>
    <mx:AddChild position="lastChild">
        <mx:Label x="210" y="37"
text="Curp:" textAlign="right"/>
    </mx:AddChild>
    <mx:AddChild position="lastChild">
        <mx:TextInput x="252" y="35"
width="120" id="txtCurp" editable="false"/>
    </mx:AddChild>
    <mx:AddChild position="lastChild">
        <mx:Label x="15" y="67"
text="Nombre:" textAlign="right"/>
    </mx:AddChild>
    <mx:AddChild position="lastChild">
        <mx:TextInput x="67" y="65"
width="117" id="txtNombre" editable="false"/>
    </mx:AddChild>
    <mx:AddChild position="lastChild">
        <mx:Label x="194" y="67"
text="Paterno:" textAlign="right"/>
    </mx:AddChild>
    <mx:AddChild position="lastChild">
        <mx:TextInput x="252" y="65"
width="120" id="txtPaterno" editable="false"/>
    </mx:AddChild>
    <mx:AddChild position="lastChild">
        <mx:Label x="380" y="67"
text="Materno:" textAlign="right"/>
    </mx:AddChild>
    <mx:AddChild position="lastChild">
        <mx:TextInput x="437" y="65"
width="120" id="txtMaterno" editable="false"/>
    </mx:AddChild>
    <mx:AddChild relativeTo="{label1}"
position="before">
        <mx:HRule y="114" left="10"
right="10" id="hrule1"/>
    </mx:AddChild>
    <mx:AddChild relativeTo="{label1}"
position="before">
        <mx:Label x="10" y="13"
text="Resultados de la busqueda" fontWeight="bold" id="label0"/>
```

```

value="98"/>
position="before">
right="10"/>
position="before">
right="10"/>
left="10" bottom="40" id="dtgResultadosAlumno">
<mx:columns>
    <mx:DataGridColumn headerText="Materia" dataField="descripcion"/>
    <mx:DataGridColumn headerText="Profesor" dataField="profesor"/>
    <mx:DataGridColumn headerText="Curso" dataField="nombre_curso"/>
    <mx:DataGridColumn headerText="Cal. Parcial" dataField="cal_parcial"/>
    <mx:DataGridColumn headerText="Cal. Global" dataField="cal_global"/>
    <mx:DataGridColumn headerText="Cal. Final" dataField="cal_final"/>
</mx:columns>
</mx:DataGrid>
<mx:AddChild>
<mx:AddChild position="lastChild">
    <mx:Image height="50" right="95"
top="39" width="40" id="imgFoto"/>
    <mx:AddChild>
        <mx:AddChild position="lastChild">
            <mx:Button label="Volver a
consultar" right="10" bottom="10">
                <mx:icon>@Embed(source='..../assets/iconos/arrow_refresh.pn
g')</mx:icon>
            </mx:Button>
        </mx:AddChild>
        <mx:RemoveChild target="(dtgResultados)"/>
    </mx:State>
    <mx:HRule y="29" left="10" right="10"/>
    <mx:Label x="10" y="13" text="Datos de busqueda"
fontWeight="bold" id="label1"/>
    <mx:TextInput y="39" left="10" right="101" id="txtData"/>
    <mx:Button y="39" label="Buscar" right="10"
id="btnBuscar"
icon="@Embed(source='..../assets/iconos/magnifier.png')"/>
    <mx:DataGrid right="10" left="10" bottom="10" top="69"
id="dtgResultados">
        <mx:columns>
            <mx:DataGridColumn headerText="Matricula"
dataField="matricula"/>
            <mx:DataGridColumn headerText="Curp"
dataField="curp"/>
            <mx:DataGridColumn headerText="Nombre"
dataField="nombre"/>
            <mx:DataGridColumn headerText="Paterno"
dataField="paterno"/>
            <mx:DataGridColumn headerText="Materno"
dataField="materno"/>
        </mx:columns>
    </mx:DataGrid>

```

Creando resultados por primer estado

Para continuar con el desarrollo de esta interfaz, generaremos nuestra primer búsqueda , ya que tenemos nuestro PHP, ahora solo queda generar nuestro HTTPService y traer al grid el resultado:

```
<mx:HTTPService id="filtroPersonalizado"
    result="filtroPersonalizadoHandler(event)"
    fault="fallaBuscarAlumnos(event)"
    url="data/buscarAlumnoFiltro.php"
    showBusyCursor="true"
    method="POST"/>
<mx:Script>
    <![CDATA[
        import mx.controls.Alert;
        import mx.rpc.events.FaultEvent;
        import mx.rpc.events.ResultEvent;
        private function filtroPersonalizadoHandler(evento:ResultEvent):void{
            dtgResultadosdataProvider = evento.result.alumnos.alumno;
        }
        private function fallaBuscarAlumnos(evento:FaultEvent):void{
            Alert.show(evento.fault.faultString, "Error");
        }
    ]]>
</mx:Script>
```

Ahora lanzamos el filtro.

```
<mx:Button y="39" label="Buscar" right="10"
    id="btnBuscar" icon="@Embed(source='../assets/iconos/magnifier.png')"
    click="filtroPersonalizado.send({d:txtDato.text})"/>
```

Ahora nuestro grid tiene la información que requerimos para encontrar. Lo que nos resta es saber que hacer con esa información.

En el listado del grid vamos a cambiar o agregar propiedad que se llama doubleClickEnabled y lo pasamos a true, ahora agregamo el doubleClick y llamamos a seleccionarAlumno()

```
<mx:DataGrid right="10"
    doubleClickEnabled="true"
    doubleClick="seleccionarAlumno(event)"
    left="10" bottom="10" top="69" id="dtgResultados">
```

Ahora creamos el método seleccionarAlumno(). Aquí lanzaremos nuestra siguiente consulta a la base de datos sobre el archivo PHP "filtroAlumnoDetalle.php".

```
<?php
$dato = $_POST['d'];
include_once 'conexion.php';
$netConex = conectar ();
$resultado = mysql_query ("SELECT DISTINCT
    grupo.id_grupo,
    materias.descripcion,
    empleados.nombre,
    empleados.paterno,
    empleados.materno,
    cursos.nombre_curso,
    alumno_calif.cal_parcial,
    alumno_calif.cal_global,
    alumno_calif.cal_final,
    alumno_calif.matricula
FROM
alumno_calif
INNER JOIN grupo ON (alumno_calif.grupo_id=grupo.id_grupo)
INNER JOIN materias ON (grupo.materia_id=materias.id_materia)
INNER JOIN empleados ON (grupo.empleado_id=empleados.rfc)
INNER JOIN cursos ON (materias.pertenece_curso=cursos.id_curso)
where alumno_calif.matricula = '$dato')";
if (mysql_affected_rows ($netConex) > 0)
{
    header ( "Content-type: text/xml" );
    print "<?xml version=\"1.0\" encoding=\"utf-8\"?>\n";
    print "<alumnos>\n";
    while ( ($row = mysql_fetch_row ( $resultado )) != false
) {
        print "<alumno>";
        print "<descripcion>" . $row [1] .
"</descripcion>\n";
        print "<profesor>" . $row [2] . ".$row[3]."
".$row[4]. "</profesor>\n";
        print "<nombre_curso>" . $row [5] .
"</nombre_curso>\n";
        print "<cal_parcial>" . $row [6] .
"</cal_parcial>\n";
    }
}
```

```

        print "<cal_global>" . $row [7] .
"</cal_global>\n";
        print "<cal_final>" . $row [8] .
"</cal_final>\n";
        print "</alumno>\n";
    }
    print "</alumnos>";

}else print "0";

?>

```

Este código recibira el dato a buscar para el filtro, el cual será la matrícula. Es tiempo de generar el `HTTPService` para el segundo filtro.

```

<mx:HTTPService id="segundoFiltro"
    result="segundoFiltroHandler(event)"
    fault="fallaCompBuscarAlumnos(event)"
    url="doo/FiltroAlumnoDetalle.php"
    showBusyCursor="true"
    method="POST"
/>

```

Y creamos nuestro método “segundoFiltroHandler()” y lanzaremos `segundoFiltro.send()` en la opción del segundo clic “seleccionarAlumno”

```

private function seleccionarAlumno(evento:MouseEvent):void{
    segundoFiltro.send({d:evento.target.data.matricula});
}

```

Y colocamos un punto de interrupción en nuestro “segundoFiltroHandler” y hacemos un debug para obtener resultados correctos. De haber estado correcto todo, pasamos al siguiente estado y mandamos llenar el grid.

```

private function segundoFiltroHandler(evento:ResultEvent):void{
    currentState="resultados";
    dtgResultadosAlumnodataProvider = evento.result.alumnos.alumno;
}

```

Solo nos queda generar nuestras variables que contendrá el nombres del alumno y sus datos.

```

[Bindable] private var matricula_c:String;
[Bindable] private var curp_c:String;
[Bindable] private var nombre_c:String;
[Bindable] private var paterno_c:String;
[Bindable] private var materno_c:String;
[Bindable] private var foto_c:String;

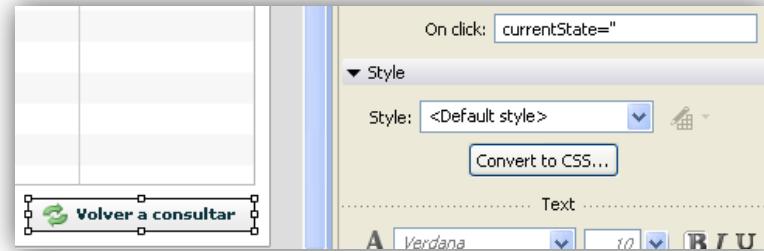
```

Y nuestro método “seleccionarAlumno” quedará de esta manera.

```
private function seleccionarAlumno(evento:MouseEvent):void{
    segundoFiltro.send({d:evento.target.data.matricula});
    matricula_c = evento.target.data.matricula;
    nombre_c = evento.target.data.nombre;
    paterno_c = evento.target.data.paterno;
    materno_c = evento.target.data.materno;
    curp_c = evento.target.data.curp;
    Foto_c = evento.target.data.foto;
}
```

Ya terminado de ralizar cambiamos el text de nuestras cajas de texto.

Resultados de la búsqueda		
Matrícula: {matricula_c}	Curp: {curp_c}	
Nombre: {nombre_c}	Paterno: {paterno_c}	Materno: {materno_c}



Y la fotografía tendrá también una variable de tipo Bindable.



Y nuestro botón para regresar al primer estado.

Registrar Alumno Inscripciones Status

Resultados de la búsqueda

Matrícula:	2009D2001	Curn:	ZAOC810304HDFRRR05		
Nombre:	CARLOS	Paterno:	ZARAGOZA	Materno:	ORTIZ

Materias del alumno

Materia	Profesor	Curso	Cal. Parcial	Cal. Global	Cal. Final
COMPRESION DEL	ALFREDO ALDAPE	DANZA	0	0	0
ACONDICIONAMIE	ALFREDO ALDAPE	DANZA	0	0	0
TECNICA MODERN	AFREDO CABRERA	DANZA	0	0	0
TECNICA CLASICA	ESMERALDA CHAV	DANZA	0	0	0
TECNICA MODERN	AMALIA CANO MEN	DANZA	0	0	0

 [Volver a consultar](#)

Reportes y graficas

Objetivos

- Instalar entorno de desarrollo
- Implementar reportes simples y con parámetros
- Crear graficas
- Personalizar contenido de graficas



INSTALANDO JDK, NETBEANS Y IREPORT

Cada uno de estos programas nos ayudaran a generar reportes, esto significa que necesitamos tener un servidor para paginas JSP, ya que usaremos Servlets para llamar a nuestros reportes.

Primero descargaremos las versiones que requerimos, estas serán:

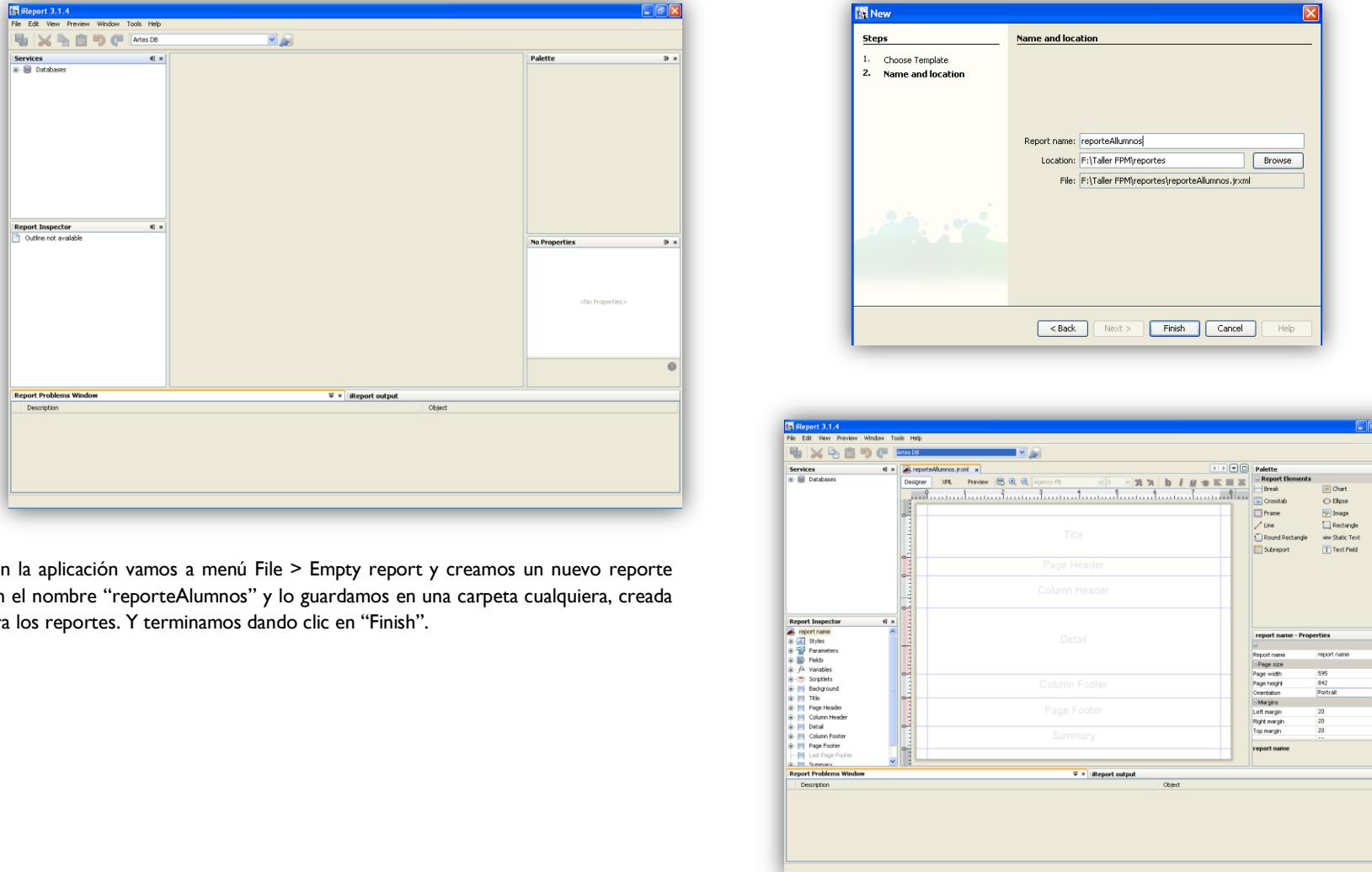
<http://java.sun.com/javase/downloads/index.jsp>

<http://www.netbeans.org/downloads/index.html>

<http://java.sun.com/javase/downloads/index.jsp>

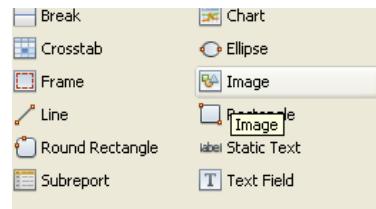
Cada uno de estos programas será necesario para generar nuestros reportes, bajamos e instalamos cada uno de ellos, empezamos con el jdk, netbeans y ireport.

Ya que se instalarón, abrimos nuestro iReport, y veamos como esta construido.



Con la aplicación vamos a menú File > Empty report y creamos un nuevo reporte con el nombre “reporteAlumnos” y lo guardamos en una carpeta cualquiera, creada para los reportes. Y terminamos dando clic en “Finish”.

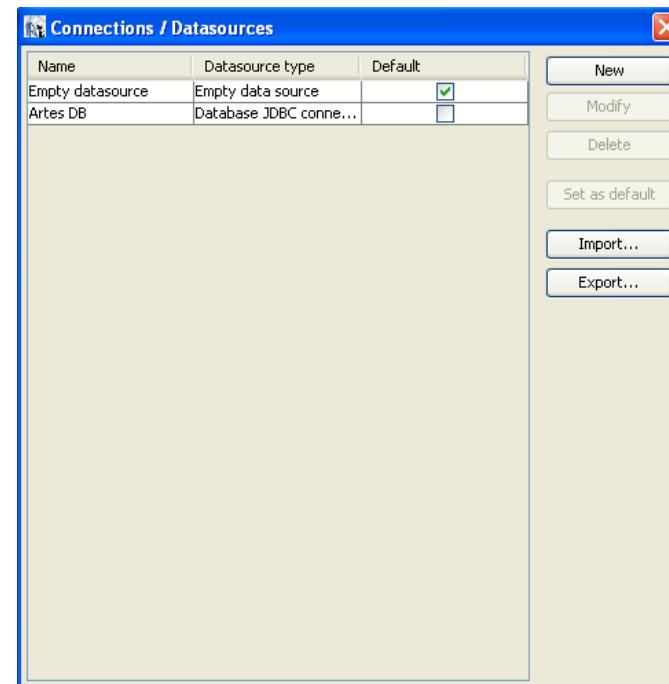
Esta es la ventana de nuestro editor lo que vamos a realizar será un encabezado usando el control image:



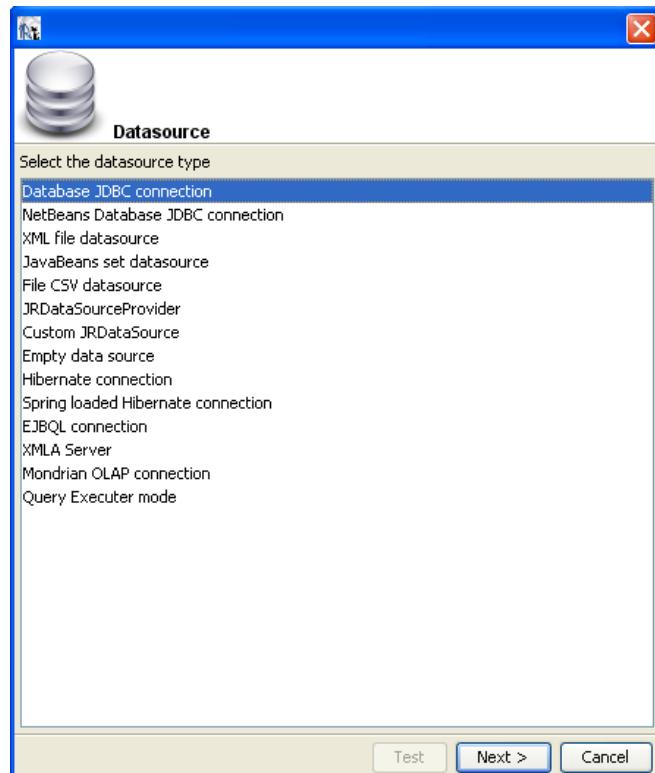
Ya dentro de nuestro editor vamos a generar imágenes, textos estaticos, los cuales nos ayudaran a tener el siguiente resultado:



Aquí generaremos un nuevo datasource, damos clic al icono de conexión:



Creamos una nueva y usamos el DataBase JDBC connection



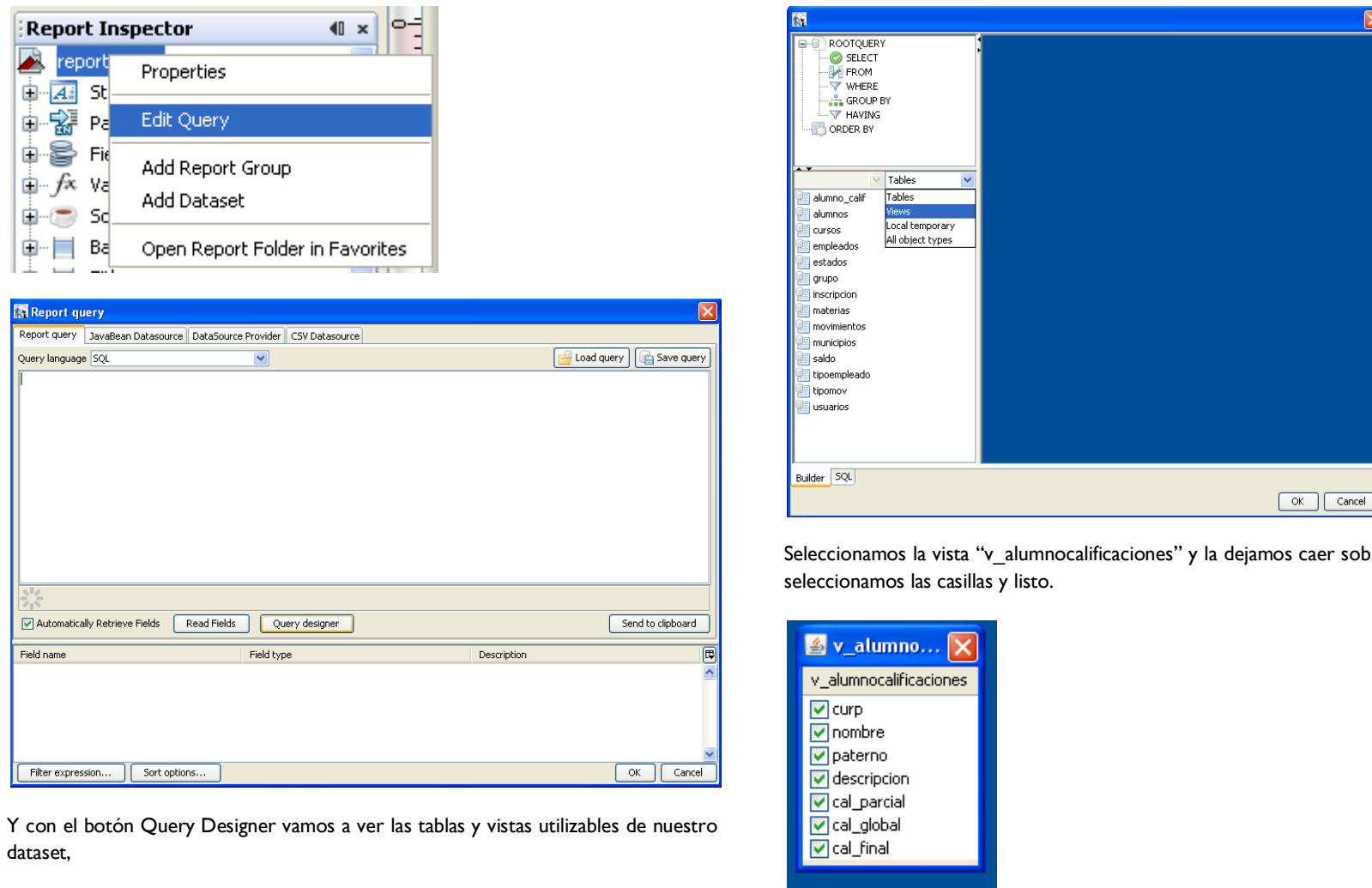
Le damos siguiente al botón y colocamos un nombre la dirección de nuestro servidor el tipo de driver así como el usuario y su password.



Probamos a conexión y listo, salvamos la conexión y ahora nuestro dataset, ya será el que acabamos de generar.

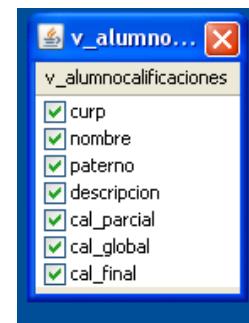


Ahora en el “Report Inspector” sobre report name damos segundo botón y add Query.

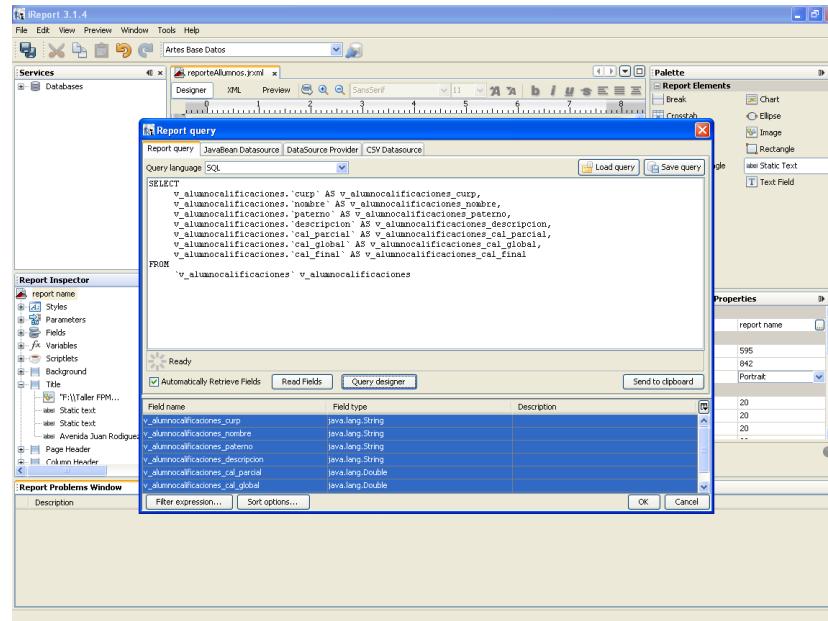


Y con el botón **Query Designer** vamos a ver las tablas y vistas utilizables de nuestro dataset,

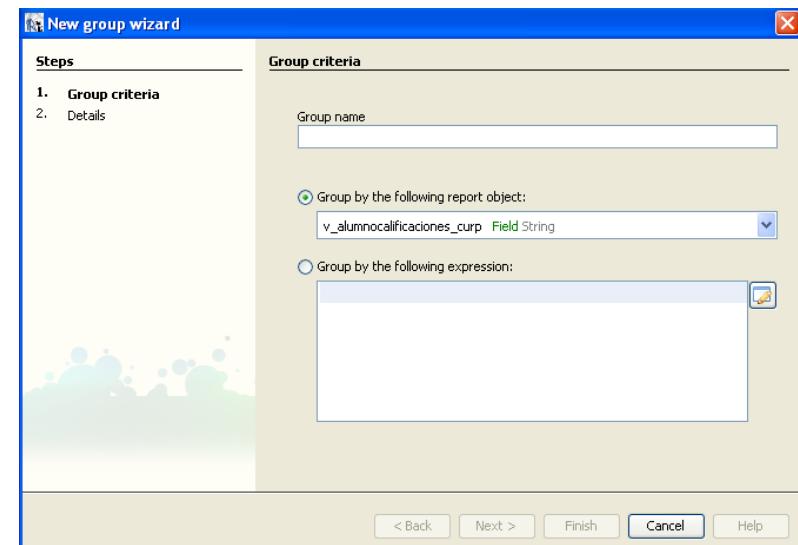
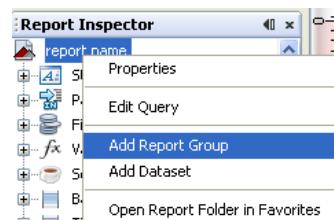
Seleccionamos la vista “v_alumnocalificaciones” y la dejamos caer sobre el area azul, seleccionamos las casillas y listo.



Si todo esta bien veremos una ventana como la siguiente:

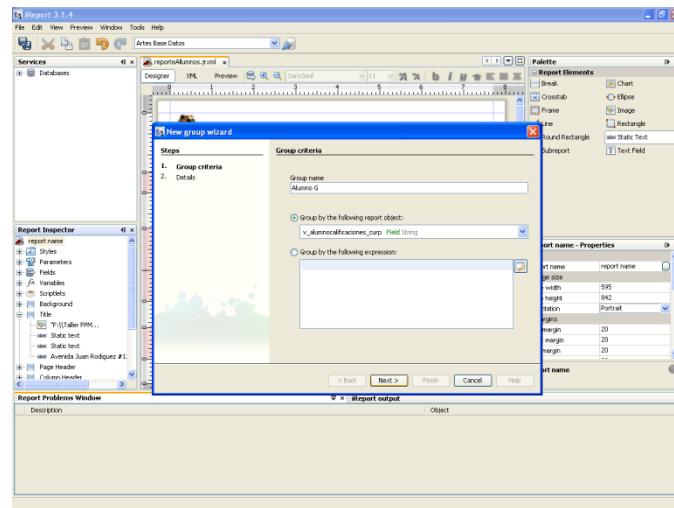


Damos clic en OK y seguimos con la edición de nuestro reporte. Todo reporte requiere de un encabezado, título, pie, y también de grupo, es decir en el caso del reporte necesitamos saber quién será el principal actor de nuestra agrupación. Para generar un nuevo en el mimos lugar donde obtuvimos el Query vamos a tener el "add report group"

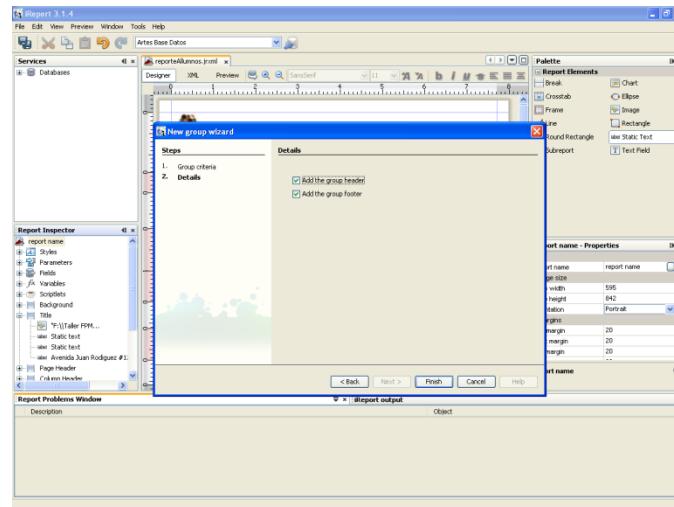


Nos pide el nombre de grupo y quien será el principal actor de agrupación, para nosotros será el CURP.

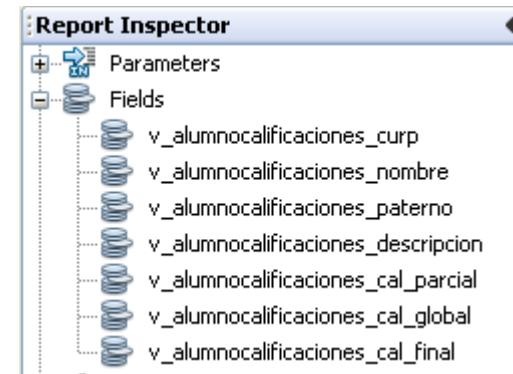
► Taller de aplicaciones RIA



Damos clic en siguiente:



Finalizamos nuestro reporte.



Ahora tenemos nuestros campos listos para ser usados en nuestro reporte así como el grupo para dividir.



En el grupo "Alumno G" vamos a agregar los campos

CURP

NOMBRE

PATERNO

Y en el detalle:

CAL_PARCIAL

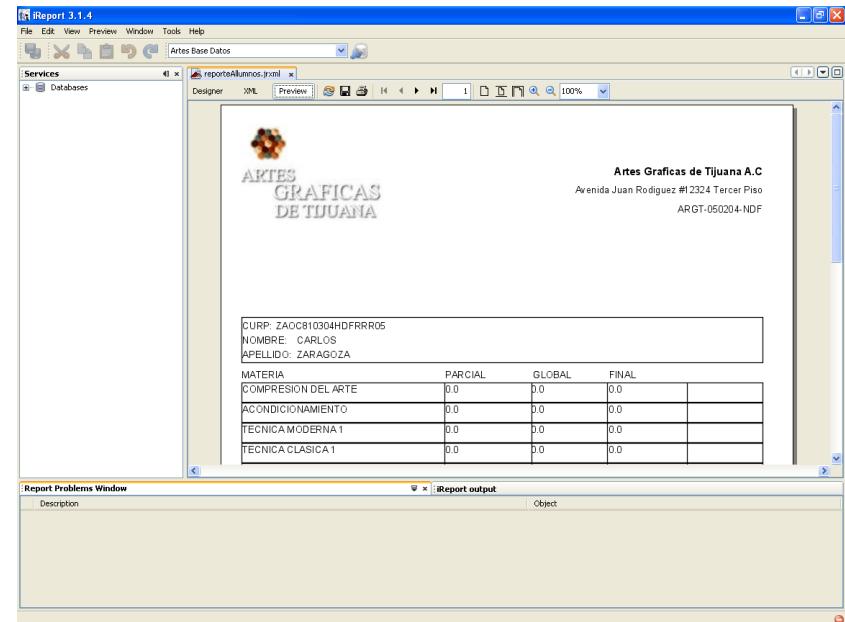
CAL_GLOBAL

CAL_FINAL

No sin antes tener el diseño de nuestro reporte, esto loaremos con las herramientas de la paleta.

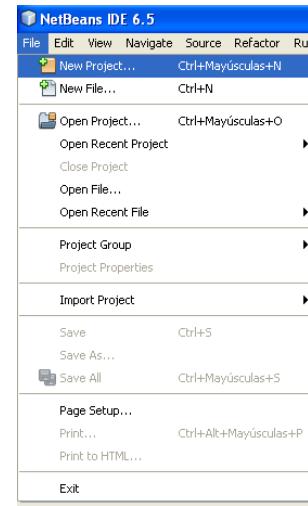
MATERIA	PARCIAL	GLOBAL	FINAL
\$F{v_alumnocalificaciones_descripcion}	\$F	\$F	\$F

Así debe quedar nuestro reporte, ahora solo es cuestión de probar nuestro reporte. Para esto usamos el botón de Preview.

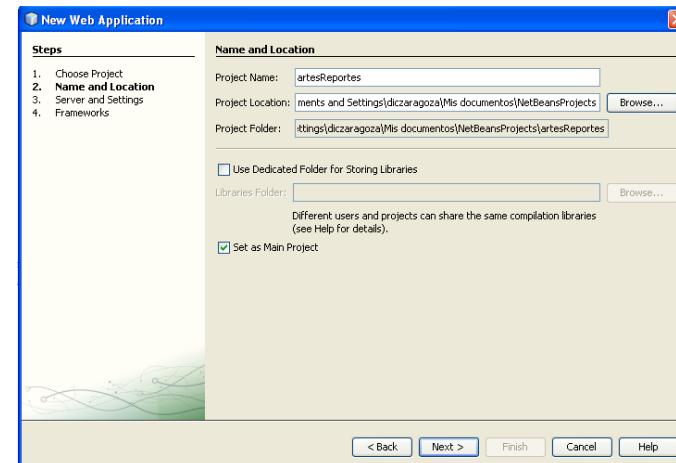


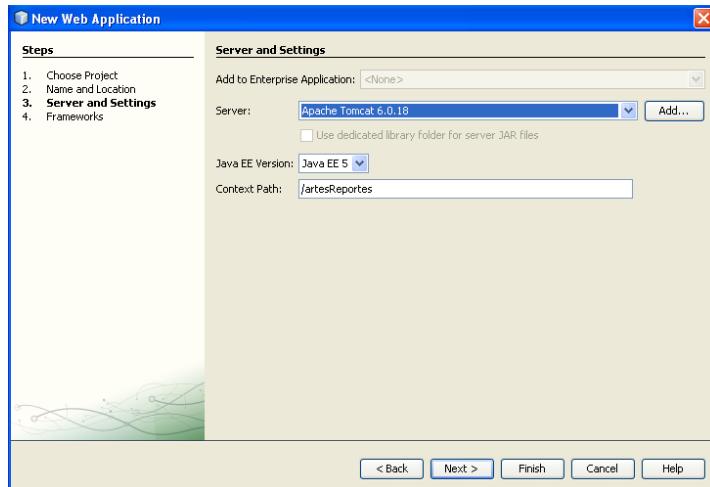
	Artes Graficas de Tijuana A.C. Avenida Juan Rodriguez #12324 Tercer Piso ARGT-090204-NDF		
REPORTE DE CALIFICACIONES			
CURP: ZAC0102100304HDFRNR03 NOMBRE: CARLOS APELLIDO: ZARAGOZA			
MATERIA	PARCIAL	GLOBAL	FINAL
COMPRESION DEL ARTE	0.0	0.0	0.0
ACONDICIONAMIENTO	0.0	0.0	0.0
TECNICA MODERNA 1	0.0	0.0	0.0
TECNICA CLASICA 1	0.0	0.0	0.0
TECNICA MODERNA EJECUTANTE 1	0.0	0.0	0.0
CURP: GAGA01040291MHGRRNAT NOMBRE: ANAH APELLIDO: GARCIA			
MATERIA	PARCIAL	GLOBAL	FINAL
COMPRESION DEL ARTE	0.0	0.0	0.0
ACONDICIONAMIENTO	0.0	0.0	0.0
TECNICA MODERNA 1	0.0	0.0	0.0
TECNICA CLASICA 1	0.0	0.0	0.0
TECNICA MODERNA EJECUTANTE 1	0.0	0.0	0.0
CURP: GAGA01040291MHGRRNAB NOMBRE: ANA APELLIDO: ASO			
MATERIA	PARCIAL	GLOBAL	FINAL
COMPRESION DEL ARTE	0.0	0.0	0.0
ACONDICIONAMIENTO	0.0	0.0	0.0
TECNICA MODERNA 1	0.0	0.0	0.0
TECNICA CLASICA 1	0.0	0.0	0.0
TECNICA MODERNA EJECUTANTE 1	0.0	0.0	0.0

Esta es la cara Final de nuestro reporte. Una vez terminado crearemos nuestro Servlet y lo agregaremos a nuestro reporte junto con las librerías.



Al generar nuestro proyecto será uno para web.





```

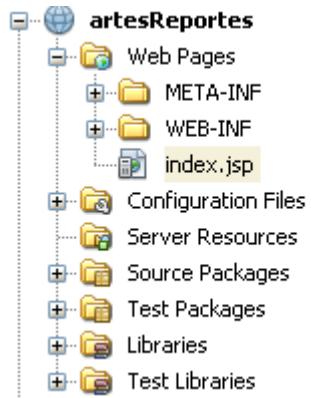
<%-->
Document : index
Created on : 27-feb-2009, 14:24:34
Author : diczaragoza

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

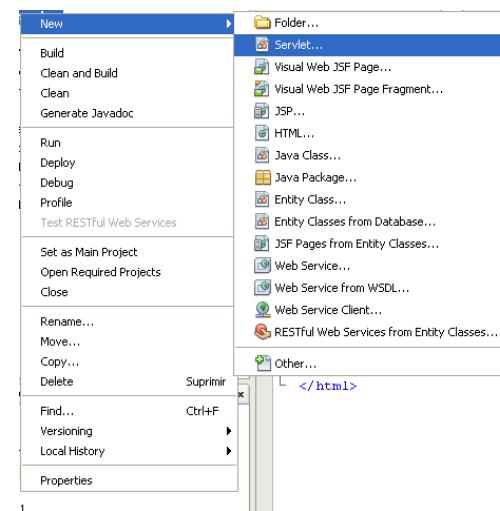
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Reportes de Artes Graficas de Tijuana A.C.</title>
</head>
<body>
<h1></h1>
</body>
</html>

```

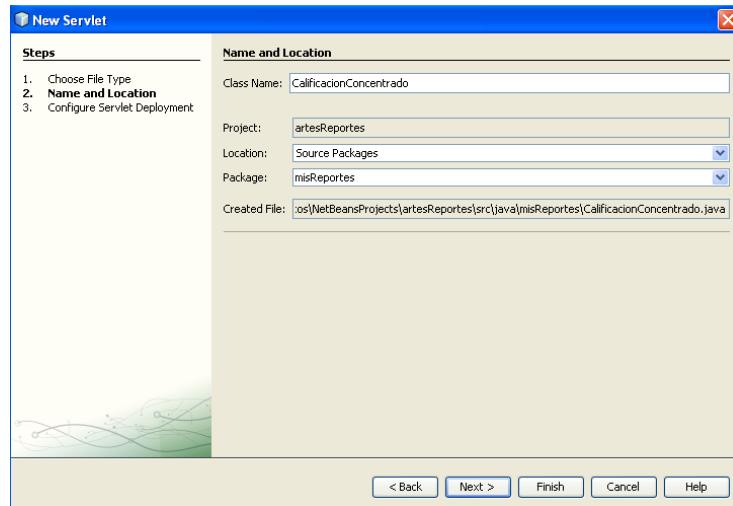
Dejamos tomcat y JavaEE 5 y terminamos el asistente



Este es el directorio que termina creando nuestra aplicación web. En el JSP, vamos a quitar el cuerpo solo dejar el head.



Ahora generamos nuestro servlet que estará trabajando con los reportes que generemos en iReports. Y finalizamos.



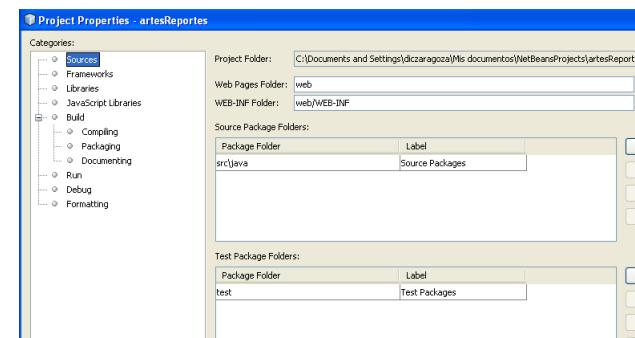
```
package misReportes;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class CalificacionConcentrado extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        try {
        } finally {
        }
    }
}
```

[HttpServlet methods. Click on the + sign on the left to edit the code.]

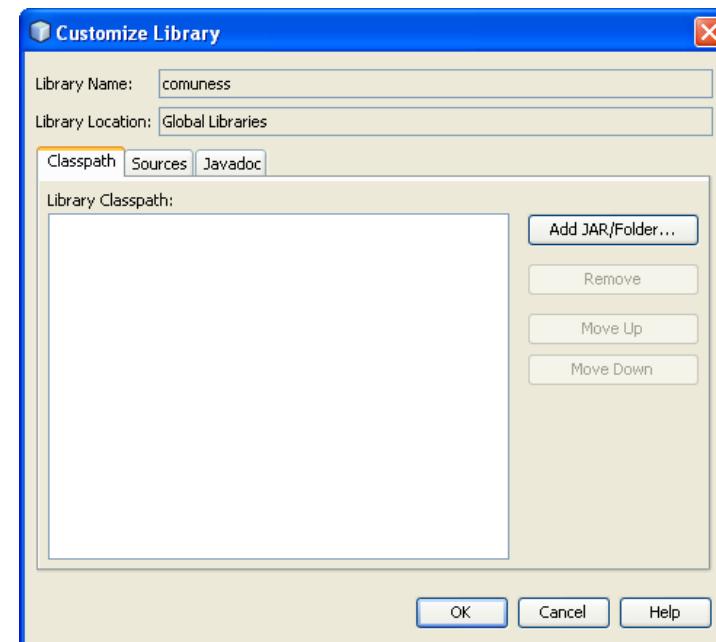
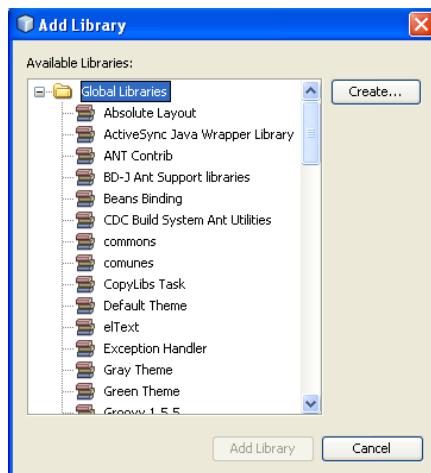
Antes de iniciar a modificar este servlet, agregamos las librerías que se usarán estas son:

- ▶ Mysql
- ▶ JasperReports
- ▶ Commons
- ▶ iText

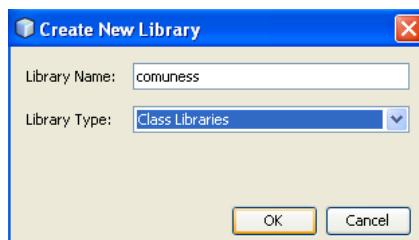
Para esto damos clic derecho sobre el proyecto en la opción propiedades,



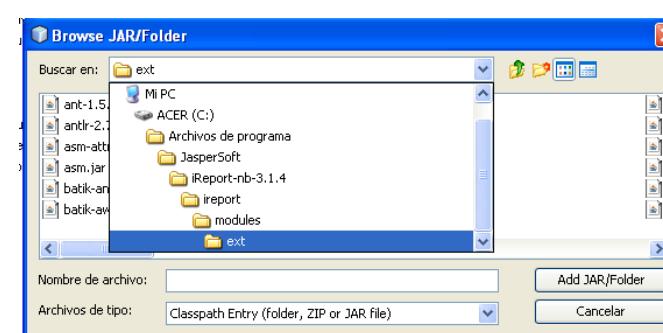
Vamos al área de librerías y utilizamos el botón add library



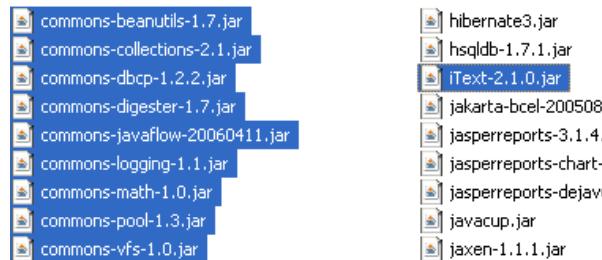
Creamos una nueva y agregamos las siguientes librerías.



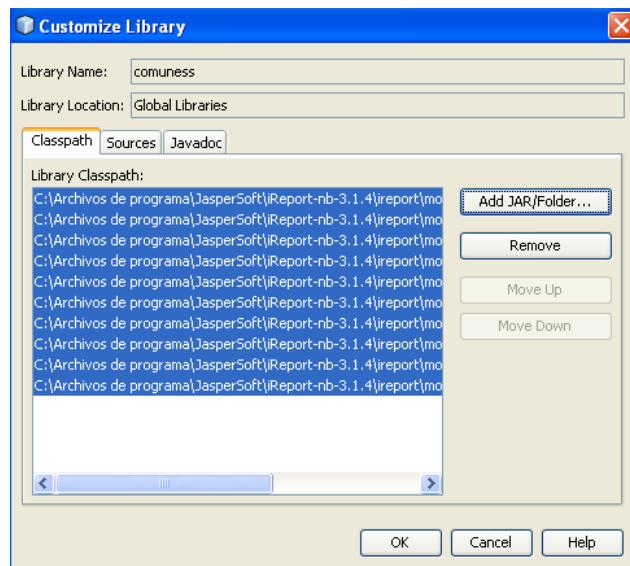
Ahora es tiempo de usar el “Add JAR/Folder”,



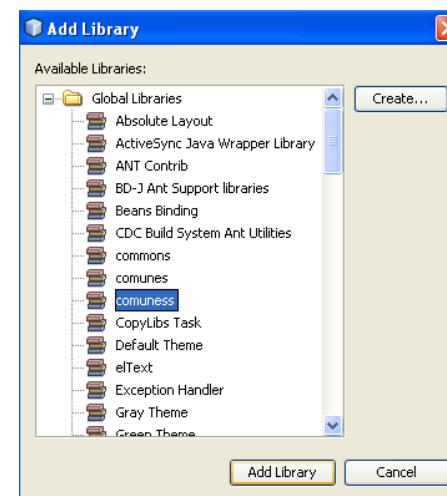
Buscamos el JAR en esta dirección y tomamos los siguientes:



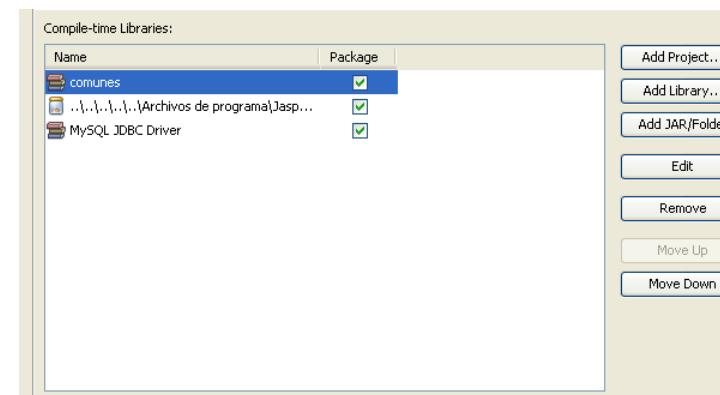
Después de agregarlos



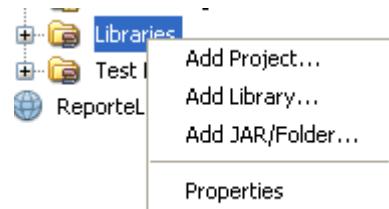
Damos un clic en OK



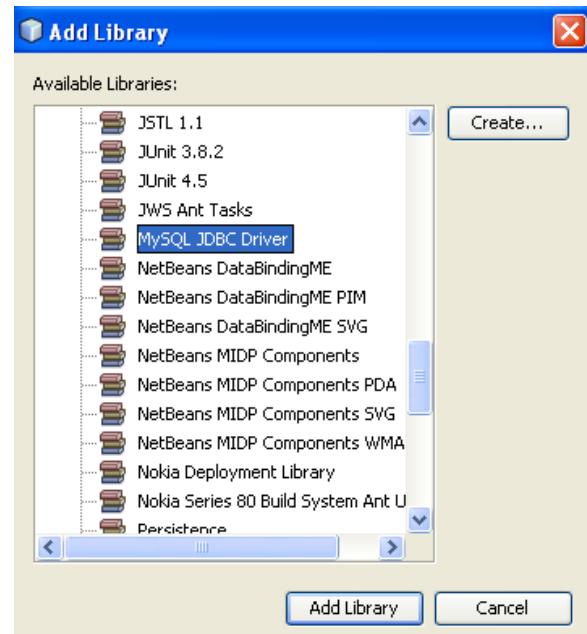
Estara lista en nuestras librerías solo damos clic en "Add Library"



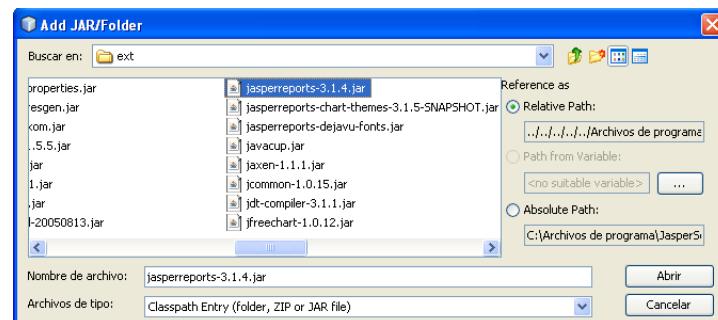
Ahora vamos sobre el proyecto en la carpeta de "Libraries"



Y agregamos una Librería, esta será la de Mysql



Damos clic a “Add Library” y solo nos faltaría agregar la librería de JasperReports esta es con la opción Add JAR/Folder.

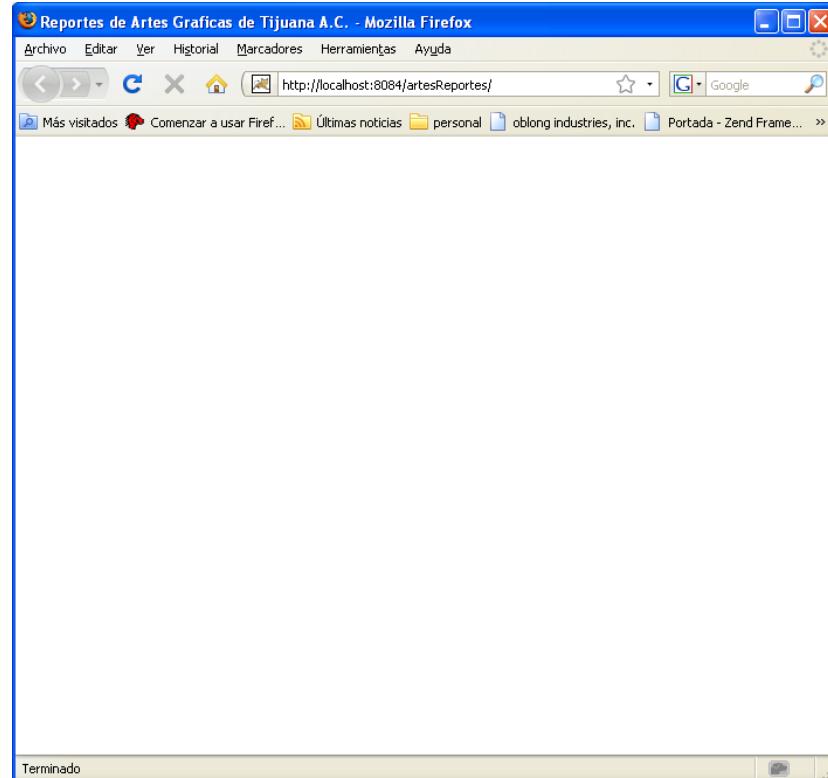


La cual está en la misma dirección que las demás librerías.

Ahora es tiempo de escribir el código, este es el cuerpo de nuestro servlet ya modificado ahí es donde empezaremos a teclear lo siguiente:

```
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/artes","root","");
OutputStream salida;
String reporteSource = "";
String base = getServletConfig().getServletContext().getRealPath("/");
reporteSource = base + "reporteAlumnos.jrxml";
Map parametros = new HashMap();
JasperReport jasper = JasperCompileManager.compileReport(reporteSource);
byte[] fichero = JasperRunManager.runReportToPdf(jasper, parametros,con);
response.setContentType("application/pdf");
response.setHeader("Content-disposition", "inline; filename=reporte.pdf");
response.setHeader("Cache-Control", "max-age=30");
response.setHeader("Pragma", "No-cache");
response.setDateHeader("Expires", 0);
response.setContentLength(fichero.length);
salida = response.getOutputStream();
salida.write(fichero, 0, fichero.length);
salida.flush();
salida.close();
```

Terminando copiamos nuestro reporte a la carpeta Web Pages. Y probamos.



This screenshot shows a Mozilla Firefox window with the title bar 'CalificacionConcentrado (application/pdf Objeto) - Mozilla Firefox'. The menu bar includes Archivo, Editar, Ver, Historial, Marcadores, Herramientas, and Ayuda. The toolbar includes Back, Forward, Stop, Home, and Search buttons. The address bar shows the URL 'http://localhost:8084/arteReportes/CalificacionConcentrado'. Below the toolbar, there are links for 'Más visitados', 'Comenzar a usar Fire...', 'Últimas noticias', 'personal', 'oblong industries, inc.', and 'Portada - Zend Frame...'. The main content area displays three separate PDF reports for different students:

- Artes Graficas de Tijuana A.C.**
Avenida Juan Rodriguez #10224 Tijuana, B.C.
ARGT-090204-ADP
- REPORTE DE CALIFICACIONES**
- CURP: ZACG070320HCRRROS**
NOMBRE: CARLOS ALBERTO ZACARIAZOS

MATERIA	PARCIAL	GLOBAL	FINAL
COMPRENSION DEL ARTE	10.0	10.0	10.0
CONDICIONAMIENTO	10.0	10.0	10.0
TECNICA MODERNA 1	10.0	10.0	10.0
TECNICA CLASICA 1	10.0	10.0	10.0
TECNICA MODERNA EJECUTANTE 1	10.0	10.0	10.0

- CURP: GAGD040204MHRRINAT**
NOMBRE: ANAH APELLEIDO: GARCIA

MATERIA	PARCIAL	GLOBAL	FINAL
COMPRENSION DEL ARTE	10.0	10.0	10.0
CONDICIONAMIENTO	10.0	10.0	10.0
TECNICA MODERNA 1	10.0	10.0	10.0
TECNICA CLASICA 1	10.0	10.0	10.0
TECNICA MODERNA EJECUTANTE 1	10.0	10.0	10.0

- CURP: GAGD040204MHRRINAS**
NOMBRE: ANA APELLEIDO: ASO

MATERIA	PARCIAL	GLOBAL	FINAL
COMPRENSION DEL ARTE	10.0	10.0	10.0
CONDICIONAMIENTO	10.0	10.0	10.0
TECNICA MODERNA 1	10.0	10.0	10.0
TECNICA CLASICA 1	10.0	10.0	10.0
TECNICA MODERNA EJECUTANTE 1	10.0	10.0	10.0

Terminamos los reportes. Si queremos usar parámetros, es necesario declararlos con:

```
params.put("curp_r", request.getParameter("curp"));
```

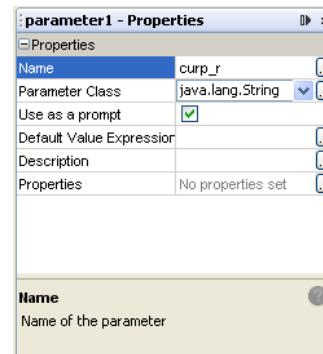
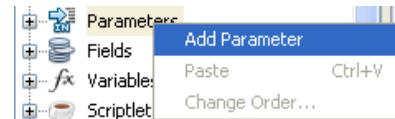
En el caso de requerir el parámetro. A modificar. Y en nuestro reporte agregamos un parámetros que se llame igual curp_r y lo usamos para filtrar nuestra consulta con la opción de llamada \${curp_r}. código:

```

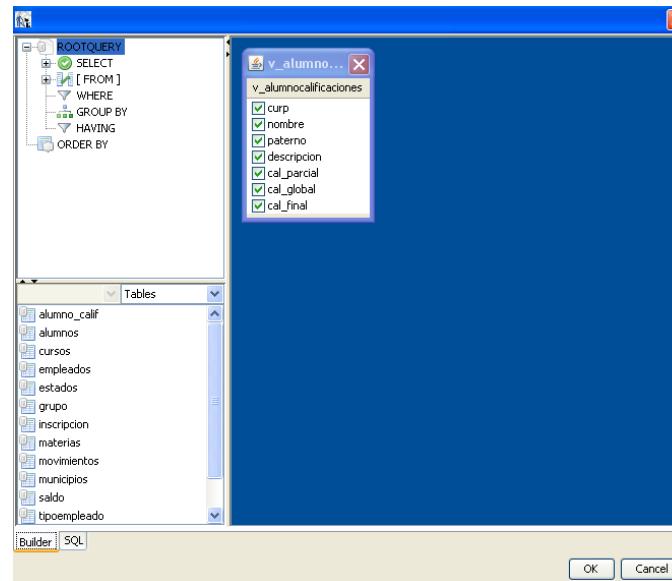
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection("jdbc:mysql://localhost");
OutputStream salida;
String reporteSource = "";
String base = getServletConfig().getServletContext().getRealPath("/");
reporteSource = base + "reporteCaliXAlum.jrxml";
Map parametros = new HashMap();
parametros.put("curp_r", request.getParameter("curp"));
JasperReport jasper = JasperCompileManager.compileReport(reporteSource);
byte[] fichero = JasperRunManager.runReportToPdf(jasper, parametros, response);
response.setContentType("application/pdf");
response.setHeader("Content-disposition", "inline; filename=repo");
response.setHeader("Cache-Control", "max-age=30");
response.setHeader("Pragma", "No-cache");
response.setDateHeader("Expires", 0);
response.setContentLength(fichero.length);
salida = response.getOutputStream();
salida.write(fichero, 0, fichero.length);
salida.flush();
salida.close();

```

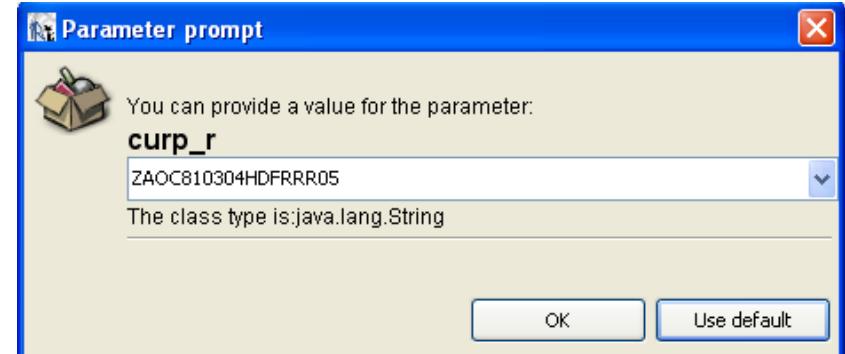
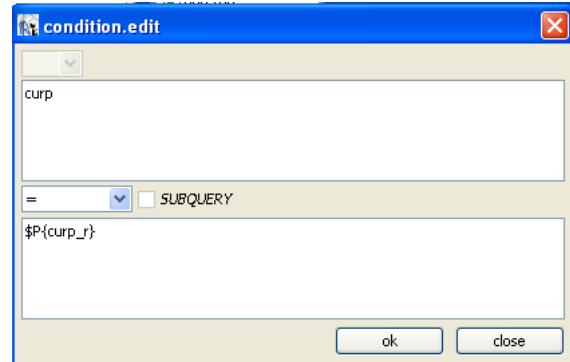
Ahora es cuestión de crear otro reporte, a este lo llamaremos “reporteCaliXAlum” y vamos a agregar un parámetro, esto es en el Inspector del reporte:



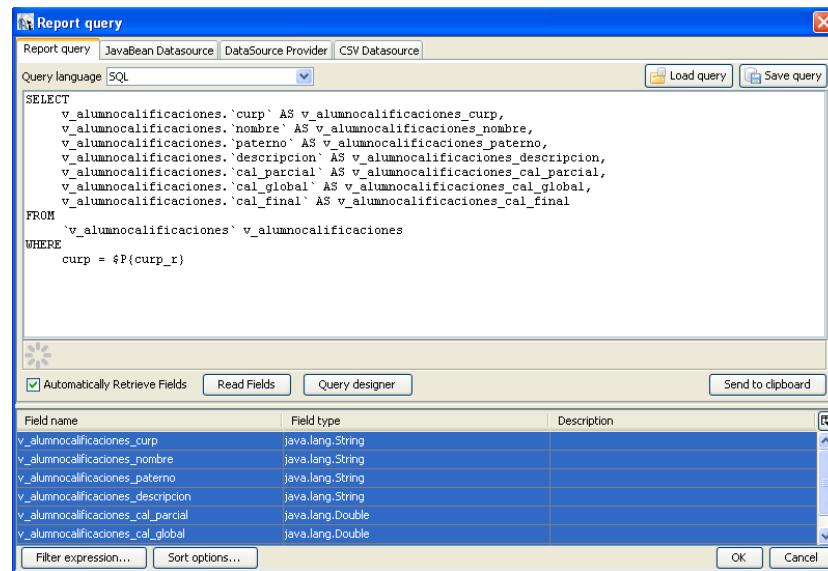
Y ahora modificamos el Query, en el Report Query> Query designer.



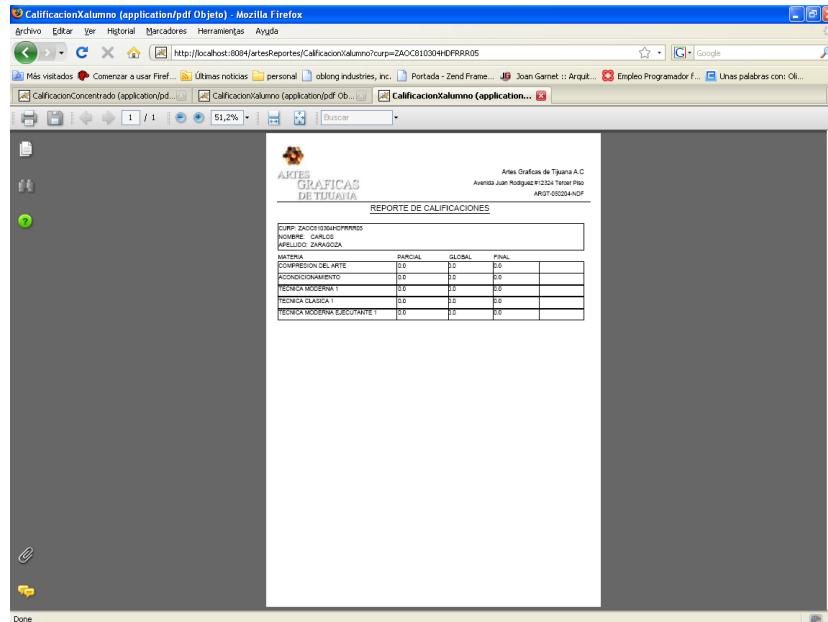
Segundo botón sobre la opción WHERE > add condition



Ahora solo aceptamos, nuestro diseñador ahora se vera de esta manera con la condicional dirigida al parámetro, ahora probamos en Preview y damos un curp valido.



Solo tendremos el resultado del parámetro solicitado. Ahora es tiempo de probarlo con el servlet. No sin antes haber copiado el reporte en el directorio Web Pages



Ya que tenemos dos reportes podemos incorporarlos al Flex para esto agregaremos un botón en la parte de debajo de nuestra primer estado del componente “`buscarAlumnos`”

Cada uno con su texto indicado y generaremos dos métodos para cada uno de los botones, el del primer estado será

- ▶ imprimirReporte()
 - ▶ imprimirReporteAlumno()

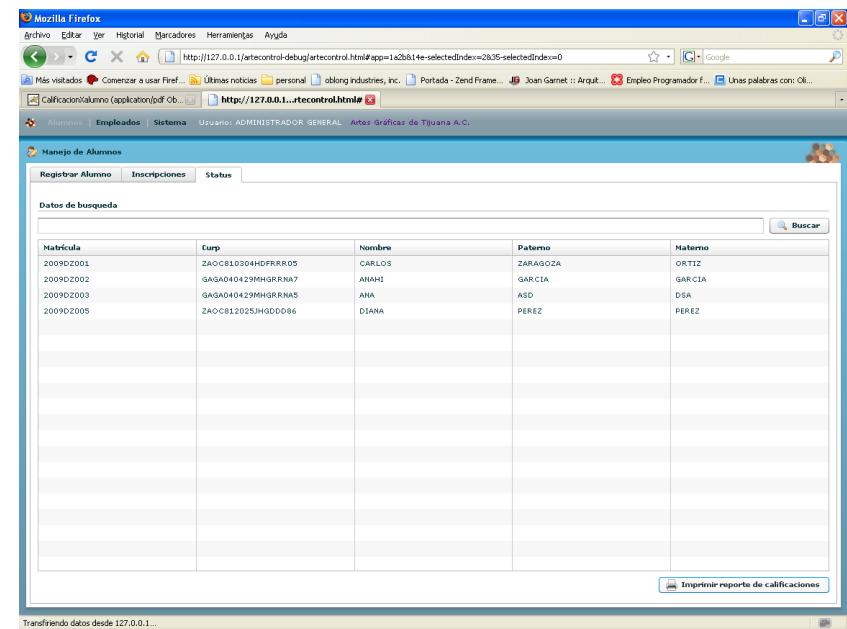
Ya que se tienen vamos a programarlos de esta manera:

```
private function imprimirReporte():void{
    var lanzarReporte:URLRequest =
    new URLRequest("http://localhost:8084/artesReportes/CalificacionConcentrado");
    try{
        navigateToURL(lanzarReporte);
    }catch(error:Error){
        Alert.show("Imposible lanzar el reporte","Error");
    }
}

private function imprimirReporteAlumno():void{

    var lanzarReporte:URLRequest =
    new URLRequest("http://localhost:8084/artesReportes/CalificacionXalumno?curp="+curp_c);
    try{
        navigateToURL(lanzarReporte);
    }catch(error:Error){
        Alert.show("Imposible lanzar el reporte","Error");
    }
}
```

Ahora solo es cuestión de probar con la dirección IP de nuestro servidor de aplicaciones.



CalificaciónConcentrado (application/pdf Objeto) - Mozilla Firefox

ArteS GRAFICAS DE TIJUANA A.C.

Avenida Juan Rodriguez 1234 Terreno 100
Méjico 22400 Tijuana, B.C. C.P.

REPORTE DE CALIFICACIONES

CURP:	ZACGEG500901PRMOS	PARCIAL	GLOBAL	FINAL
NOMBRE:	JOSE CARLOS			
APPELLIDO:	ZAMORA			
MATERIA:	COMPRESION DEL ARTE	0.0	0.0	0.0
	CONDICIONAMIENTO	0.0	0.0	0.0
	TECNICA MODERNA 1	0.0	0.0	0.0
	TECNICA CLASICA 1	0.0	0.0	0.0
	TECNICA MODERNA EJECUTANTE 1	0.0	0.0	0.0

CURP:	GAGA040429MHGRRNAT	PARCIAL	GLOBAL	FINAL
NOMBRE:	ANAHÍ			
APPELLIDO:	GARCIA			
MATERIA:	COMPRESION DEL ARTE	0.0	0.0	0.0
	CONDICIONAMIENTO	0.0	0.0	0.0
	TECNICA MODERNA 1	0.0	0.0	0.0
	TECNICA CLASICA 1	0.0	0.0	0.0
	TECNICA MODERNA EJECUTANTE 1	0.0	0.0	0.0

CURP:	GAGA040429MHGRRNAT	PARCIAL	GLOBAL	FINAL
NOMBRE:	AMALIA CANO MENDOZA			
APPELLIDO:	ASD			
MATERIA:	COMPRESION DEL ARTE	0.0	0.0	0.0
	CONDICIONAMIENTO	0.0	0.0	0.0
	TECNICA MODERNA 1	0.0	0.0	0.0
	TECNICA CLASICA 1	0.0	0.0	0.0
	TECNICA MODERNA EJECUTANTE 1	0.0	0.0	0.0

CalificaciónAlumno (application/pdf Objeto) - Mozilla Firefox

ArteS GRAFICAS DE TIJUANA A.C.

Manejo de Alumnos

Registrar Alumno Inscripciones Status

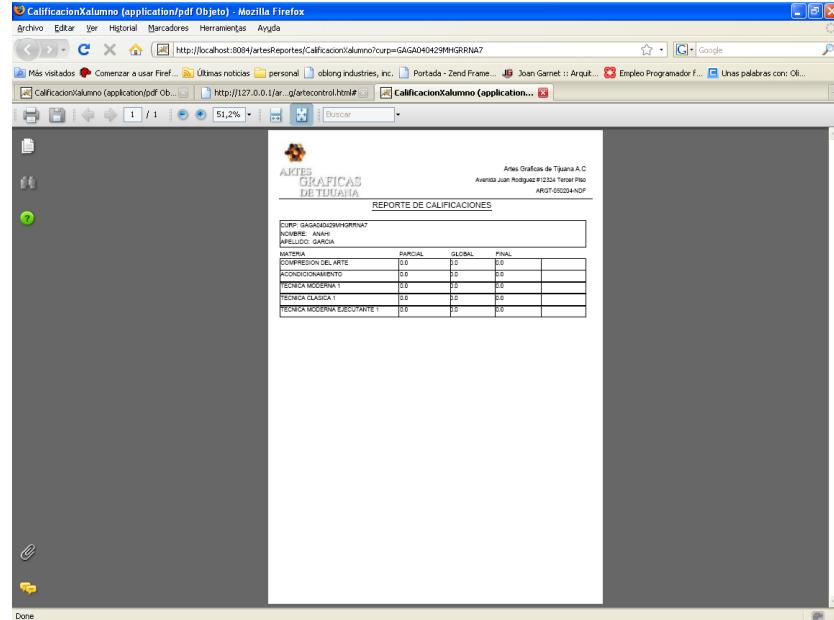
Resultados de la búsqueda

Matrícula:	209902002	Curp:	GAGA040429MHGRRNAT
Nombre:	ANAHÍ	Paterno:	GARCIA
Materno:	GARCIA		

Materias del alumno

Materia	Profesor	Curso	Cal. Parcial	Cal. Global	Cal. Final
COMPRESION DEL ARTE	ALFREDO ALDAPE ACUÑA	DANZA	0	0	0
CONDICIONAMIENTO	ALFREDO ALDAPE ACUÑA	DANZA	0	0	0
TECNICA MODERNA 1	ALFREDO CABRERA HERNANDEZ	DANZA	0	0	0
TECNICA CLASICA 1	ESMERALDA CHAVEZ HERNANDEZ	DANZA	0	0	0
TECNICA MODERNA EJECUTANTE 1	AMALIA CANO MENDOZA	DANZA	0	0	0

[Imprimir calificaciones](#) [Volver a consultar](#)



Graficas en Flex

Todos los graficos en flex, tienen atributos básicos, esto son:

- ▶ Chart: define el tipo de grafica, se especifica el set de datos y los estilos
- ▶ Series: define los datos mostrados en las series sobre el grafico, coloca parámetros como líneas, fills y renderes. Por default usa los datos del tag del chart.
- ▶ Axes: son implementados por default a excepción de los graficos de Pie, los controles de despliegue son Labels y estos se definen horizontal y verticalmente.
- ▶ Elements: opcional define una cuadricula como elemento de visión secundaria.

Las propiedades básicas del componente son: id, width y height. En las series es una serie de arreglos basados en objetos.

Dataprovider

ShowDataTips es un booleano que define si se despliegan los tips o no.

Creando un Chart

1. Hay que especificar una etiqueta de chart.
2. Agregar las siguientes propiedades
 - a. Id
 - b. Width
 - c. Height
 - d. Dataprovider
 - e. Showdatatips
3. Opcionalmente se definen los ejes
4. Se agregan las series
5. Dentro de las series se agregan un array
6. Cada elemento dentro del array deberá ser una específica serie para el chart. (ColumnSeries/LineSeries/PieSeries).

```
<mx:ColumnChart id="[instance name of chart]" 
height="[pixel or percent]" 
width="[pixel or percent]" 
dataProvider="[array collection to chart]">
```

```
showDataTips="[true|false]">
<mx:series>
<mx:Array>
<mx:ColumnSeries
displayName="[label for series]"
xField="[name of field in dataProvider]"
yField="[name field in dataProvider]" />
</mx:Array>
</mx:series>
</mx:ColumnChart>
```

Table 2: *Basic ColumnSeries properties*

PROPERTY	DESCRIPTION
displayName	User display for the name of the series within the DataTips
xField	Specifies which field in the dataProvider to use for the x-axis location (optional)
yField	Specifies which field in the dataProvider to use for the height of the individual columns

Vamos a crear ahora una grafica con las siguientes caracteristicas:

Primero generamos nuestra fuente de datos con un XML este podrá ser creado con cualquier otro lenguaje.

```
<resultados>
    <meses>
        <mes name="Enero">
            <ingreso>156541</ingreso>
            <egreso>7537</egreso>
        </mes>
        <mes name="Febrero">
            <ingreso>645481</ingreso>
            <egreso>753357</egreso>
        </mes>
        <mes name="Marzo">
            <ingreso>45481</ingreso>
            <egreso>456654</egreso>
        </mes>
```

```
<mes name="Abril">
    <ingreso>81545</ingreso>
    <egreso>66445</egreso>
</mes>
<mes name="Mayo">
    <ingreso>645481</ingreso>
    <egreso>645481</egreso>
</mes>
<mes name="Junio">
    <ingreso>664455</ingreso>
    <egreso>667768</egreso>
</mes>
<mes name="Julio">
    <ingreso>66581</ingreso>
    <egreso>99889</egreso>
</mes>
<mes name="Agosto">
    <ingreso>87787</ingreso>
    <egreso>988888</egreso>
</mes>
<mes name="Septiembre">
    <ingreso>351245</ingreso>
    <egreso>645481</egreso>
</mes>
<mes name="Octubre">
    <ingreso>87877</ingreso>
    <egreso>645481</egreso>
</mes>
<mes name="Noviembre">
    <ingreso>998877</ingreso>
    <egreso>645481</egreso>
</mes>
<mes name="Diciembre">
    <ingreso>3332214</ingreso>
    <egreso>645481</egreso>
</mes>
</meses>
</resultados>
```

Este es un ejemplo de ingresos y egresos de cada mes. En una nueva aplicación vamos a enlace a nuestros datos por medio de un **HTTPService**.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            import mx.controls.Alert;
            import mx.rpc.events.FaultEvent;
            import mx.rpc.events.ResultEvent;
            private var resultados:ArrayCollection = new ArrayCollection();
            private function enlaceHandler(evento:ResultEvent):void{
                resultados = evento.result.resultados.meses.mes;
            }
            private function fallaEnlace(evento:FaultEvent):void{
                Alert.show(evento.fault.faultString,"Error");
            }
        ]]>
    </mx:Script>
    <mx:HTTPService id="enlace" result="enlaceHandler(event)" fault="fallaEnlace(event)" url="data/resultado.xml"/>
</mx:Application>
```

Ahora es tiempo de agregar nuestra grafica:

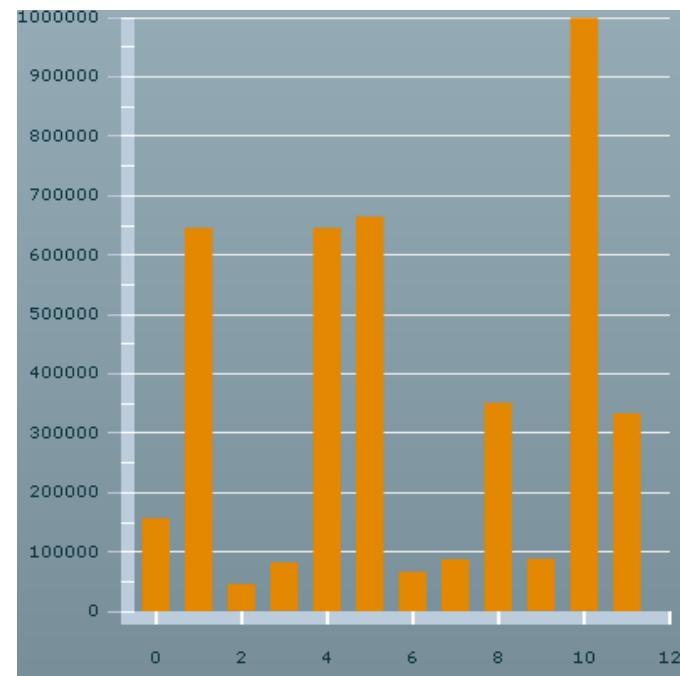
```
<mx:CartesianChart
    id="miGrafico"
    dataProvider="{resultados}">

</mx:CartesianChart>
```

Ya que se ha creado el cuerpo, ahora hay que generar las series que contendrán nuestros datos a mostrar.

```
<mx:series>
    <mx:ColumnSeries displayName="Ingreso" yField="ingreso"/>
</mx:series>
```

En este caso usamos un ColumnSeries, donde mostraremos el nombre y el campo del arrayCollection.



Hasta este momento no hay nada especial en estas graficas, lo que nos falta es darle formato a los ejes. Crearemos un método para generar el formato de moneda en el eje horizontal. Primero un control currencyFormat

```
<mx:CurrencyFormatter id="cf" precision="2" currencySymbol="$"/>
```

Después creamos una función que nos reciba los valores de nuestra verticalAxis que se encargara de aplicar el formato y regresar un String.

```
private function FormatoMoneda(valor:Number, valorPrev:Number,eje:Object):String{  
    return cf.format(valor);  
}
```

Por último generamos un verticalAxis.

```
<mx:verticalAxis>  
    <mx:LinearAxis labelFunction="FormatoMoneda" />  
</mx:verticalAxis>
```

Nuestro grafico ahora tendrá esta forma.

```
<mx:CartesianChart  
    id="miGrafico"  
    dataProvider="{resultados}">  
    <mx:verticalAxis>  
        <mx:LinearAxis labelFunction="FormatoMoneda" />  
    </mx:verticalAxis>  
    <mx:series>  
        <mx:ColumnSeries displayName="Ingreso" yField="ingreso"/>  
    </mx:series>  
</mx:CartesianChart>
```



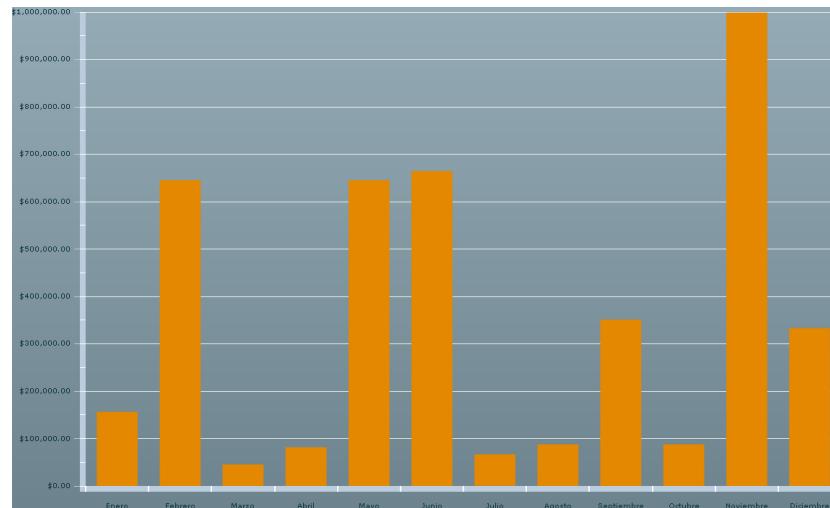
Ahora necesitamos ver el nombre de cada barra, este será el horizontalAxis sobre el categoryField. Recodemos que el categoryField es el atributo del nodo.

```
<mx:horizontalAxis>  
    <mx:CategoryAxis categoryField="name" />  
</mx:horizontalAxis>
```

Nuestro grafico quedaría ahora de esta manera.

```
<mx:CartesianChart
    id="miGrafico"
    dataProvider="{resultados}">
    <mx:verticalAxis>
        <mx:LinearAxis labelFunction="FormatoMoneda" />
    </mx:verticalAxis>
    <mx:horizontalAxis>
        <mx:CategoryAxis categoryField="name"/>
    </mx:horizontalAxis>
    <mx:series>
        <mx:ColumnSeries displayName="Ingreso" yField="ingreso"/>
    </mx:series>
</mx:CartesianChart>
```

Ahora ajustamos en tamaño de nuestro grafico. 100% en width y height.



Hasta este momento podemos tener nuestra grafica lista para ser visualizada, ahora podemos detallarla con efectos y tips sobre cada resultado.

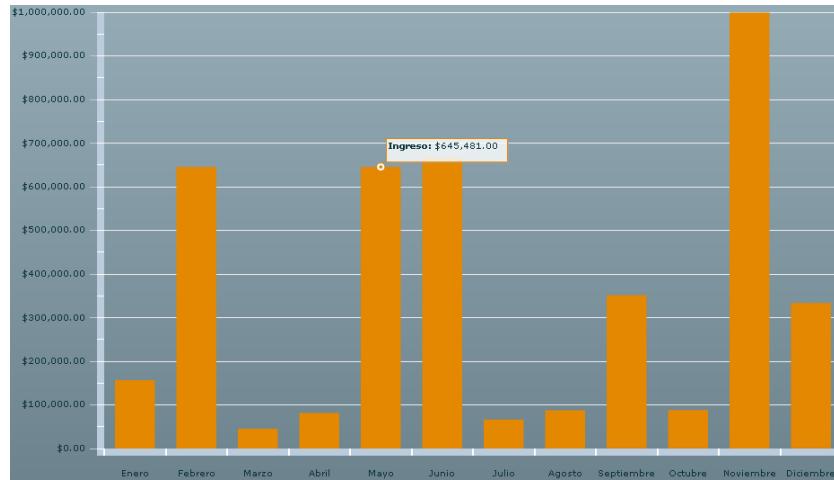
Para los DataTips que serian como los tooltips, podemos crear una función que haga el trabajo de mostrar el detalle de cada elemento,

```
private function miDataTip(mi_hit:HitData):String{
    var sRegreso:String= "";
    sRegreso += "<b>Ingreso: </b>" + cf.Format(mi_hit.item.ingreso)+"<br>";
    return sRegreso;
}
```

Ahora esta función se agregara al grafico en la opción dataTipFunction.

```
<mx:CartesianChart width="100%" height="100%"
    dataTipFunction="miDataTip"
    id="miGrafico"
    showDataTips="true"
    dataProvider="{resultados}">
    <mx:verticalAxis>
        <mx:LinearAxis labelFunction="FormatoMoneda" />
    </mx:verticalAxis>
    <mx:horizontalAxis>
        <mx:CategoryAxis categoryField="name"/>
    </mx:horizontalAxis>
    <mx:series>
        <mx:ColumnSeries displayName="Ingreso" yField="ingreso"/>
    </mx:series>
</mx:CartesianChart>
```

El resultado de esto ahora podemos verlo.



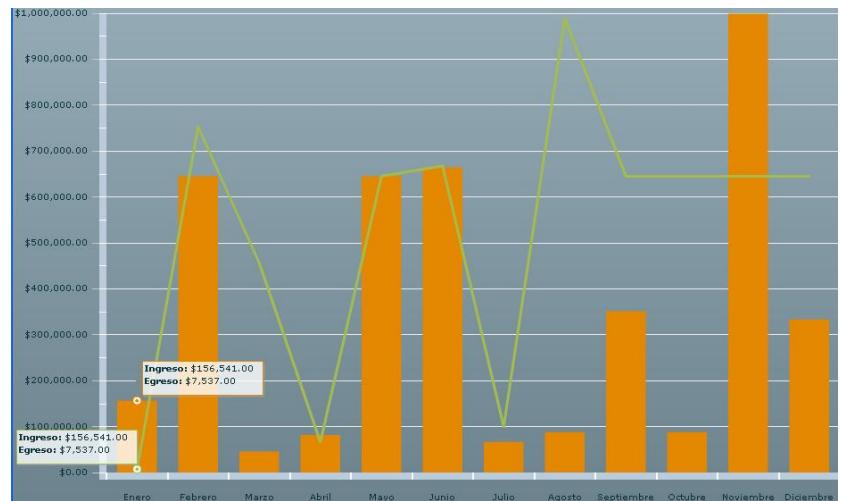
Ahora crearemos efectos para que aparezcan los graficos.

```
<mx:SeriesInterpolate id="efxInterpolate" duration="1000"/>
<mx:ColumnSeries showDataEffect="efxInterpolate"
.
```

Con esto podemos ver animado nuestro grafico, si requerimos mostrar más de un dato para nuestra aplicación, podemos usar dentro de ColumnSeries otro tipo de graficos o de los mismo basados en CartesianChart.

Ejemplo:

```
<mx:CartesianChart width="100%" height="100%"
dataTipFunction="miDataTip"
id="miGrafico"
showDataTips="true"
dataProvider="{resultados}">
<mx:verticalAxis>
<mx:LinearAxis labelFunction="FormatoMoneda" />
</mx:verticalAxis>
<mx:horizontalAxis>
<mx:CategoryAxis categoryField="name"/>
</mx:horizontalAxis>
<mx:series>
<mx:ColumnSeries showDataEffect="efxInterpolate" displayName="Ingreso" yField="ingreso"/>
<mx:LineSeries showDataEffect="efxSlide" displayName="Egreso" yField="egreso"/>
</mx:series>
</mx:CartesianChart>
<mx:SeriesInterpolate id="efxInterpolate" duration="1000"/>
<mx:SeriesSlide id="efxSlide" duration="1000" offset="400"/>
```



Ahora trabajo de laboratorio, será generar una grafica a partir de la consulta de los siguientes alumnos:

```

1 <?php
2 include_once 'conexion.php';
3 $netConex = conectar ();
4 $resultado = mysql_query ("SELECT DISTINCT
5     grupo.id_grupo,
6     materias.descripcion,
7     COUNT(alumno_calif.matricula) AS Alumnos
8 FROM
9     alumno_calif
.0 INNER JOIN grupo ON (alumno_calif.grupo_id=grupo.id_grupo)
.1 INNER JOIN materias ON (grupo.materia_id=materias.id_materia)
.2 INNER JOIN empleados ON (grupo.empleado_id=empleados.rfc)
.3 INNER JOIN cursos ON (materias.pertenece_curso=cursos.id_curso)
.4 GROUP BY
.5     grupo.id_grupo,
.6     materias.descripcion
.7 ");
.8 if (mysql_affected_rows ( $netConex ) > 0)
.9 {
10     header ( "Content-type: text/xml" );
11     print "<?xml version=\"1.0\" encoding=\"utf-8\"?>\n";
12     print "<resultados>\n";
13     print "<grupos>\n";
14     while ( ($row = mysql_fetch_row ( $resultado )) != False ) {
15         print "<grupo name=\"$row [1]\">\n";
16         print "<nombre>" . $row [1] . "</nombre>\n";
17         print "<cantidad>" . $row [2] . "</cantidad>\n";
18         print "</grupo>\n";
19     }
20     print "</grupos>\n";
21     print "</resultados>";
22
23 }else print "no";
24
25 ?>

```

Para saber cuántos alumnos hay en cada grupo. Y con esta información generar una grafica de los alumnos en cada grupo.

```

<mx:Script>
<!CDATA[
    import mx.utils.StringUtil;
    import mx.charts.HitData;
    import mx.collections.ArrayCollection;
    import mx.controls.Alert;
    import mx.rpc.events.FaultEvent;
    import mx.rpc.events.ResultEvent;
[Bindable] private var resultados:ArrayCollection = new ArrayCollection();
private function enlaceHandler(evento:ResultEvent):void{
    resultados = evento.result.resultados.grupos.grupo;
}
private function fallaEnlace(evento:FaultEvent):void{
    Alert.show(evento.fault.faultString,"Error");
}

private function miDataTip(mi_hit:HitData):String{
    var sRegreso:String= "";
    sRegreso += "<b>Cantidad de Alumnos: </b>" + mi_hit.item.cantidad+<br>;
    sRegreso += "<b>Materia: </b>" + mi_hit.item.nombre+<br>;
}

return sRegreso;
}
]]>
</mx:Script>

```

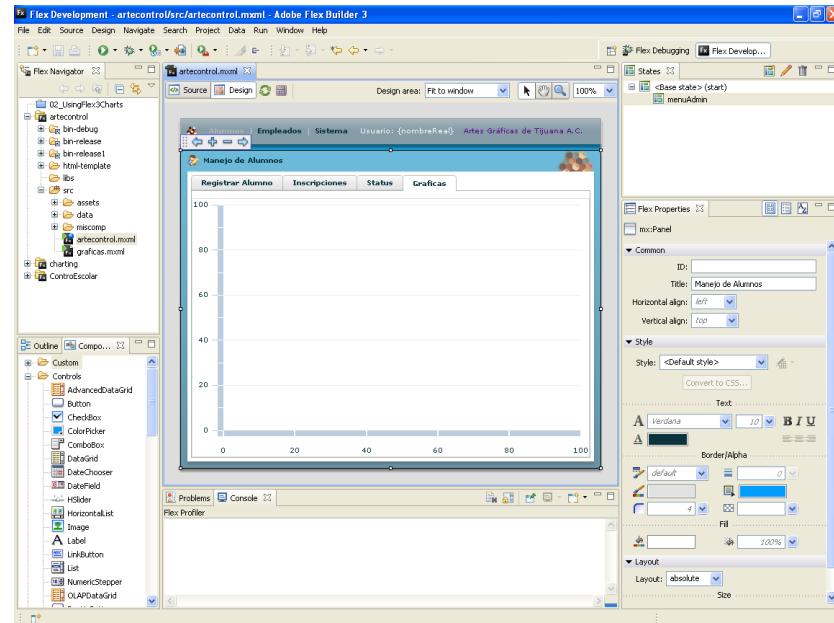
```

<mx:HTTPService id="enlace" result="enlaceHandler(event)"
  fault="fallaEnlace(event)" url="data/grafica.php"/>
<mx:CartesianChart width="100%" height="100%"
  dataTipFunction="m1DataTip"
  id="m1Grafico"
  showDataTips="true"
 dataProvider="{resultados}">

<mx:horizontalAxis>
  <mx:CategoryAxis categoryField="name"/>
</mx:horizontalAxis>
<mx:series>
  <mx:ColumnSeries showDataEffect="efxInterpolate" displayName="Alumnos" yField="cantidad"/>
</mx:series>
</mx:CartesianChart>
<mx:SeriesInterpolate id="efxInterpolate" duration="1000"/>
<mx:SeriesSlide id="efxSlide" duration="1000" offset="400"/>

```

Ahora hay que agregar este componente llamado gAlumno a nuestro Tab.



Y probar nuestra aplicación.

