PROYECTO FINAL SQL CODERHOUSE

Modelo de negocio para fábrica de guitarras

URL al repositorio del proyecto: *https://github.com/AaajFar/Proyecto-Final-SQL-CoderHouse*

El modelo de negocio es para una fábrica de guitarras, donde principalmente se encargue de la parte de stock de insumos, stock de productos (guitarras) y pedidos. La idea es que sea simple y efectivo, tratando de automatizar la mayor cantidad de procesos posibles.

- La parte de stock de materiales va a estar controlado por una tabla, con campos de marca, tipo de producto, stock, etc. Tendremos un listado de proveedores con información de los mismos con relación a la tabla de materiales a través de una tercera tabla de materiales_proveedores donde indicamos que material me provee cada uno.

- La parte de control de stock de productos (guitarras) va a estar controlado por una tabla. Va a contener información como el modelo, año, stock y precio, y va a tener relación con la tabla de materiales a través de una tercera tabla de materiales_guitarras donde voy a indicar que material se usa para cada guitarra. También va a brindar información de que guitarra se está pidiendo a la tabla pedidos.

- Los pedidos van a estar controlados por 4 tablas. La principal sería la de pedidos la cual va a contener información de clientes, precios finales, fechas y direcciones de envío. Con la tabla detalle_pedidos vamos a poder indicar que producto y cantidad lleva cada pedido, también tendremos el precio por unidad de cada detalle los cual nos va a ayudar a tener el precio final en la tabla pedidos. La tabla clientes va a contener toda la información de los mismos como nombre, apellido, teléfono, y le va a brindar la información de que cliente hizo el pedido a la tabla pedidos. La tabla direcciones va a contener información de domicilios de envío de la cartera de clientes, esta tabla recibe información de que a que cliente corresponde el domicilio y se la brinda a la tabla pedidos.

- Vamos a tener 2 tablas bitácoras sin relación con el resto de las tablas las cuales van a guardar los movimientos que realicen las tablas pedidos y materiales que son de suma importancia para la base de datos.

TABLAS

Campos/proveedores	Tipo de dato	Características	Descripción de tabla
id	INT	NOT NULL PK AI	Almacenamiento de datos de proveedores de maderas, telefono, mail, dirección.
nombre	VARCHAR(60)	NOT NULL	
dirección	VARCHAR(100)	NOT NULL	
telefono	INT	NOT NULL	
mail	VARCHAR(60)	NOT NULL	

Campos/materiales	Tipo de dato	Características	Descripción de tabla
id	INT	NOT NULL PK AI	Almacenamiento de datos de materiales, marca, modelo, tipo, stock. El campo tipo dentro del ENUM solo acepta los valores 'clavijas', 'maderas', 'microfonos' o 'cuerdas'.
tipo	ENUM	NOT NULL	
marca	VARCHAR(60)	(Default NULL)	
modelo	VARCHAR(60)	NOT NULL	
descripcion	VARCHAR(100)	(Default NULL)	
stock	INT	NOT NULL UN	
costo	FLOAT	NOT NULL	

Campos/proveedores_materiales	Tipo de dato	Características	Descripción de tabla
mat_id	INT	NOT NULL PK FK	Tabla de relación entre proveedores y materiales. Ambos campos son PK y FK de las respectivas tablas con sus id.
prov_id	INT	NOT NULL PK FK	

Campos/guitarras	Tipo de dato	Características	Descripción de tabla
id	INT	NOT NULL PK AI	Almacenamiento de datos y stock de guitarras para la venta con información de tipo, año, modelo y precio. El campo tipo dentro del ENUM solo acepta los valores 'electrica', 'clasica', 'electroacustica', 'clasica nino' o 'clasica concierto'.
tipo	ENUM	NOT NULL	
modelo	VARCHAR(60)	NOT NULL	
ano	YEAR	NOT NULL	
stock	INT	NOT NULL UN	
precio	FLOAT	(Default 0)	

Campos/materiales_guitarras	Tipo de dato	Características	Descripción de tabla
mat_id	INT	NOT NULL PK FK	Tabla de relación entre materiales y guitarras Los campos mat_id y guit_id son PK y FK de las respectivas tablas de relación con sus id.
guit_id	INT	NOT NULL PK FK	
cantidad	INT	NOT NULL	

Campos/clientes	Tipo de dato	Características	Descripción de tabla
id	INT	NOT NULL PK AI	Almacenamiento de datos de clientes, nombre, apellido, telefono, mail.
nombre	VARCHAR(45)	NOT NULL	
apellido	VARCHAR(45)	NOT NULL	
direccion	VARCHAR(200)	(Default NULL)	
telefono	INT	NOT NULL	
mail	VARCHAR(60)	NOT NULL	

Campos/direcciones_envío	Tipo de dato	Características	Descripción de tabla
id	INT	NOT NULL PK AI	Almacenamiento de direcciones de envío para los pedidos de clientes. El campo cliente_dir esta referenciado por FK al campo id_cl de la tabla clientes
cliente_id	INT	NOT NULL FK	
direccion	VARCHAR(200)	NOT NULL	
localidad	VARCHAR(60)	NOT NULL	
provincia	VARCHAR(30)	NOT NULL	

Campos/pedidos	Tipo de dato	Características	Descripción de tabla
id	INT	NOT NULL PK AI	Almacenamiento de pedidos, información de cliente, precio total, fecha y dirección de envío. El campo cliente_id esta referenciado por FK al campo id de la tabla clientes. El campo direnvio_id esta referenciado por FK al campo id de la tabla direcciones_envio.
cliente_id	INT	NOT NULL FK	
precio_total	FLOAT	(Default 0)	
fecha	DATE	NOT NULL	
direnvio_id	INT	NOT NULL FK	

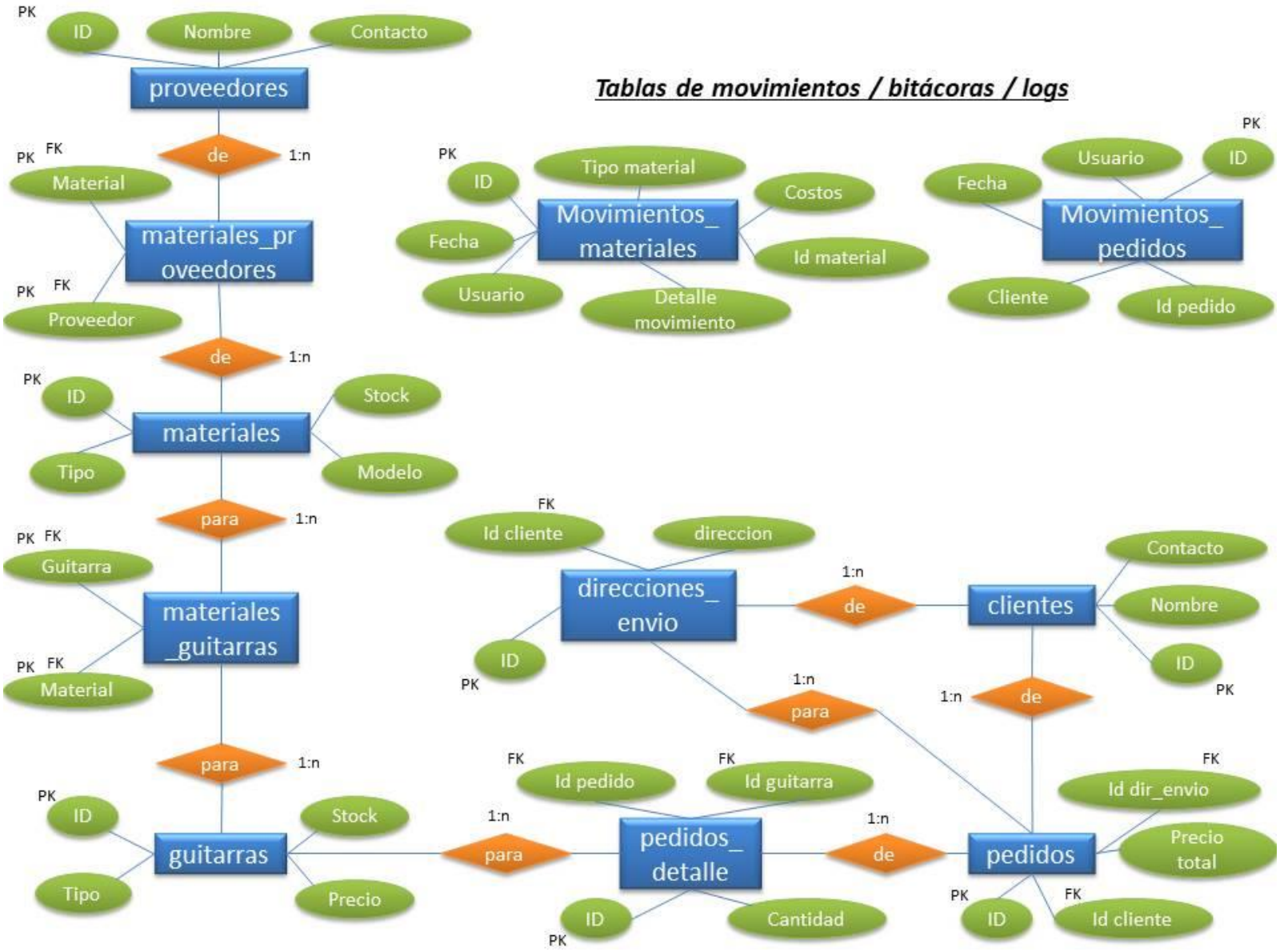
Campos/pedidos_detalle	Tipo de dato	Características	Descripción de tabla
id	INT	NOT NULL PK AI	Almacenamiento de detalle de pedidos, información de cantidad de articulos, el precio por unidad para cada id de pedidos. El campo ped_id esta referenciado por FK al campo id de la tabla pedidos. El campo guit_id esta referenciado por FK al campo id de la tabla guitarras.
ped_id	INT	NOT NULL PK	
guit_id	INT	NOT NULL PK	
guit_cant	INT	NOT NULL	
guit_precio	FLOAT	NOT NULL	

TABLAS DE MOVIMIENTOS / BITÁCORA / LOGS

Campos/movimientos_pedidos	Tipo de dato	Características	Descripción de tabla
id_mov	INT	NOT NULL PK AI	Tabla de movimientos para la tabla pedidos. Guarda los movimientos que se hayan realizado cuando se ingresa o borra un pedido con datos de fecha, hora y usuario que realizo el movimiento ademas del id, cliente y precio del pedido.
fecha_mov	DATE	NOT NULL	
hora_mov	TIME	NOT NULL	
usuario_mov	VARCHAR(50)	NOT NULL	
id_ped	INT	NOT NULL	
cliente	VARCHAR(100)	NOT NULL	

Campos/movimientos_materiales	Tipo de dato	Características	Descripción de tabla
id_mov	INT	NOT NULL PK AI	Tabla de movimientos para la tabla materiales. Guarda los movimientos que se hayan realizado cuando se ingresa o modifica el precio de un material con datos de fecha, hora y usuario que realizo el movimiento ademas del id, tipo y costo del material. En el caso de un ingreso solo guarda el costo ingresado, en el caso de una modificacion de precio guarda el precio nuevo, el precio viejo y la diferencia en porcentaje (%) entre ambos. En la columna detalle también podemos verificar si el movimiento fue un ingreso o una modificacion.
fecha_mov	DATE	NOT NULL	
hora_mov	TIME	NOT NULL	
usuario_mov	VARCHAR(50)	NOT NULL	
id_mat	INT	NOT NULL	
tipo_mat	ENUM	NOT NULL	
costo_mat	INT	NOT NULL	
costo_viejo	INT		
costo_dif	VARCHAR(10)		
detalle	VARCHAR(30)		

DIAGRAMA DE ENTIDAD-RELACIÓN



LISTADO DE VISTAS

- Vista #1 “vs_costo_guitarras”: Sirve para saber el costo de fabricación de todas las guitarras. El resultado que arroja es la sumatoria de los costos de los materiales (esto lo logramos con la función SUM de MySQL) y el id de cada una de las guitarras gracias al filtro GROUP BY que hace que se agrupen por id. Interactúa con las tablas materiales, materiales_guitarras y guitarras y es de gran utilidad ya que haciendo filtros podríamos obtener el costo de una sola guitarra o determinadas guitarras que queramos saber.

- Vista #2 “vs_total_ventas”: Sirve para saber el valor total de mis ventas (esto lo logramos gracias a la función SUM de MySQL) y en que cantidad de pedidos (esto lo logramos gracias a la función COUNT de MySQL). Interactúa solamente con la tabla pedidos haciendo el conteo de id de pedidos (cantidad de pedidos en que se vendió) y la sumatoria de los precios (total vendido).

- Vista #3 “vs_proveedores_maderas”: Sirve para obtener los datos importantes de los proveedores del tipo de material "maderas". A través de los INNER JOIN podemos combinar la información usando el filtro WHERE para que solo nos traiga información de los proveedores del tipo de material maderas, y con el GROUP BY los agrupamos para no tener información repetida. Interactúa con las tablas proveedores, proveedores_materiales y materiales, de gran utilidad para obtener una vista rápida de mis proveedores para materiales que son de los más utilizados en la fábrica.

- Vista #4 “vs_pedidos_clientes”: Sirve para ver todos los pedidos junto a su valor y dirección de envío. Interactúa con las tablas pedidos, clientes y direcciones_envio, dando un vistazo rápido al pedido, su valor y su dirección de envío, esto lo logramos gracias a los INNER JOIN que nos trae información de las tablas que necesitamos. De gran utilidad para poder identificar a donde hay que enviar cada pedido, también se puede utilizar el filtro WHERE para filtrar por el id_pedido y obtener esta misma información de uno o más pedidos determinados.

- Vista #5 “vs_material_p_guitarra”: Sirve para saber que materiales necesito para fabricar cada guitarra. Interactúa con las tablas materiales, materiales_guitarras y guitarras obteniendo el id de la guitarra, el tipo, y los materiales que la componen con sus respectivos tipos e id, esto lo logramos gracias a los INNER JOIN que nos trae información de las tablas que necesitamos. Es de gran utilidad ya que si necesito saber que materiales necesito para fabricar una determinada guitarra puedo utilizar el filtro WHERE con la vista pasándole el id_guitarra para que solo me devuelva esos datos, o bien podríamos usar el filtro WHERE con el id del material para saber que guitarras utilizan determinado material.

- Vista #6 “vs_stock_bajo_materiales”: Sirve para saber de qué materiales tengo stock bajo junto con los datos del proveedor que lo vende. Interactúa con las tablas materiales, proveedores_materiales y proveedores, con los INNER JOIN matcheamos las tablas de las cuales necesitamos traer la información, y gracias al filtro WHERE le indicamos que el stock del material tiene que ser menor o igual a 500 para que solo nos traiga los resultados inferiores a ese número. Es de gran utilidad para hacer reposición de materiales ya que podemos saber de forma rápida cual es el stock bajo y quien es el que lo provee de estos materiales, también con el filtro WHERE podríamos traer stock bajo de determinados tipos de materiales o de determinados proveedores.

- Vista #7 “vs_stock_bajo_guitarras”: Sirve para saber de qué guitarras tengo stock bajo para fabricar más. Interactúa solamente con la tabla guitarras, dando un vistazo rápido de las guitarras que tienen menos de 200 unidades de stock con su id, tipo y modelo, esto lo logramos gracias al filtro WHERE indicándole que el stock tiene que ser menor o igual a 200 para que solo nos traiga los resultados inferiores a ese número. Con el filtro WHERE podríamos traer stock bajo de guitarras de determinado tipo que es un detalle interesante si por ejemplo solo quisiéramos saber de qué guitarras eléctricas tenemos stock bajo.

- Vista #8 “vs_promedio_ventas”: Sirve para saber cuál es el promedio de ventas y en qué cantidad de pedidos. Interactúa solamente con la tabla pedidos, esto lo logramos gracias a la función AVG en el caso del promedio de ventas y a la función COUNT en el caso de cantidad de pedidos, una forma rápida de tener un vistazo del promedio de ventas.

LISTADO DE FUNCIONES

- Función #1 “fx_calc_cst_ped”: Sirve para calcular el costo de un pedido mediante el id del mismo. Declaramos la variable v_resultado la cual va a guardar la sumatoria de los costos de los materiales según el id de pedido que se ingrese al llamar a la FX multiplicándolo por la cantidad del pedido, dando como resultado el costo total para ese pedido, esto con ayuda de los INNER JOIN que nos traen información de las tablas que necesitamos y el filtro WHERE al cual le indicamos que el id del pedido tiene que ser el mismo que ingresamos al llamar a la FX. Interactúa con las tablas materiales, materiales_guitarras, guitarras y pedidos_detalle. Es de gran utilidad a la hora de ingresar un pedido nuevo para saber cuál sería el costo de fabricación del mismo.

- Función #2 “fx_calc_cst_guit”: Sirve para calcular el costo de fabricación de una guitarra mediante el id de la misma. Declaramos la variable v_resultado la cual va a guardar la sumatoria de los costos de materiales según el id de guitarra y devuelve el total, esto lo logramos gracias a la función SUM para sumar los costos, los INNER JOIN para traer la información necesaria, y el filtro WHERE donde indicamos que el id de la guitarra tiene que ser igual al id ingresado al llamar al SP. Interactúa con las tablas materiales, materiales_guitarras y guitarras, de gran utilidad para saber el costo de una guitarra de forma rápida, bien podríamos multiplicar la función por el número de guitarras que queremos fabricar para obtener el costo total del proceso. También nos será muy útil mas adelante para el Stored Procedure de actualización de costo de un material para poder actualizar también el nuevo precio de venta de una guitarra en base al nuevo costo del material.

- Función #3 “fx_calc_ctped_cl”: Sirve para calcular cuántos pedidos realizo un cliente mediante el id del mismo. Declaramos la variable v_resultado que va a guardar el resultado del conteo de cantidad de pedidos, esto lo logramos gracias a la función COUNT donde como parámetro le pasamos el id que ingresamos al llamar al SP y al filtro WHERE donde le indicamos que el id del cliente tiene que ser igual también al id que ingresamos. Puede ser útil para sacar estadísticas de quienes son los clientes que más compras realizan y en base a eso definir de qué forma pueden pagar o si tienen algún tipo de descuento.

- Función #4 “fx_calc_ctdi_ultped: Sirve para calcular cuántos días pasaron desde el ultimo pedido que hizo un cliente mediante su id. Con la variable v_fecha1 guardamos la última fecha en la que realizó un pedido según el id que se ingrese al llamar a la FX, esto lo logramos gracias al ORDER BY de manera descendente y limitando los

resultados a 1 en la consulta, con la variable v_fecha2 guardamos la fecha actual gracias al método CURDATE, luego en la variable v_resultado sacamos la diferencia entre estas dos para obtener el total de días gracias al método DATEDIFF. Interactúa solamente con la tabla pedidos y es de gran utilidad para saber si por ejemplo paso mucho tiempo desde que un cliente no realiza un pedido.

- Función #5 “fx_calc_ctped_xfech”: Sirve para calcular la cantidad de pedidos entre 2 fechas que ingrese el usuario. La función recibe como parámetros las dos fechas entre las cuales queremos saber el total, declaramos una variable v_resultado done vamos a guardar esta diferencia, luego le damos valor haciendo una consulta con el método COUNT haciendo un conteo del campo id de la tabla pedidos y con el filtro WHERE decimos que la fecha tiene que estar entre las dos ingresadas al llamar a la FX. Es de gran utilidad para saber cuántos pedidos se tuvo en un determinado periodo de forma rápida.

LISTADO DE STORED PROCEDURES

- Stored Procedure #1 “sp_orden_guitarras”: Sirve para ordenar la tabla de guitarras según los parámetros que ingresa el usuario. En total recibe 2 parámetros, el primero corresponde a sobre qué campo se va a hacer el orden, el segundo corresponde a la forma en que se va a hacer ese orden (ASC, DESC). Para poder realizarlo utilizamos el condicional CASE para cada caso, de tal forma que cuando se llama al SP los parámetros que espera son de tipo INT, siendo que en el caso del campo son valores del 1 al 6 y son en el mismo orden que esta la tabla, y en el caso del orden son valores 1 para ASC y 2 para DESC. Por ejemplo si hiciéramos un CALL con parámetros 3 y 2 lo ordenaría por modelo de forma descendente. Interactúa con la tabla guitarras, es beneficiosa para poder hacer consultas rápidas en una tabla que va a ser muy utilizada.

- Stored Procedure #2 “sp_eliminar_pedido”: Sirve para eliminar registros en la tabla pedidos, su funcionamiento es muy simple, al llamar al SP se ingresa el id del pedido que se desea eliminar y este es comparado con el id de pedido de la tabla pedidos_detalle, busca los iguales y los elimina, una vez hecho esto hace lo mismo con la tabla pedidos, primero tiene que eliminar el detalle del pedido ya que por la FK que tienen no podría eliminar el pedido si no elimino primero el detalle. Interactúa con las tablas pedidos y pedidos_detalle, es una forma rápida y fácil de eliminar registros en una tabla con mucha interacción, es normal que algún pedido pueda ser cancelado o se quiera eliminar un pedido viejo.

- Stored Procedure #3 “sp_ingresar_cliente”: Sirve para ingresar nuevos clientes, cuando se lo llama se ingresan los parámetros de nombre, apellido, dirección, teléfono y mail y crea un nuevo registro en la tabla de clientes con el próximo id. Una forma rápida y simple de ingresar un nuevo cliente.

- Stored Procedure #4 “sp_actstock_guitarra”: Sirve para ingresar stock de las guitarras fabricadas, su gran ventaja es que también hace una actualización del stock de los materiales utilizados para la fabricación. Al ser llamado se ingresan como parámetros el id de la guitarra que se desea actualizar y el stock que le queremos agregar. Declaramos tres variables, v_idmat, v_cantmat y v_finbucle que nos van a ayudar a guardar la información generada ya que a continuación declaramos un CURSOR que va a hacer un SELECT del id del material y cantidad de la tabla materiales_guitarras donde el id de la guitarra sea igual al id ingresado al llamar al SP, luego con un loop va

a recorrer todos los materiales cuyo id de guitarra coincida. Declaramos un CONTINUE HANDLER para NOT FOUND y le decimos que actualice la variable v_finbucle a 1 (su DEFAULT es 0, funcionaría como TRUE or FALSE), esto nos va a ayudar a poder parar el loop ya que sino seria infinito. Declaramos un EXIT HANDLER para el error 1690 (BIGINT UNSIGNED) ya que el campo stock de la tabla materiales no puede ser negativo por lo tanto si no disponemos stock suficiente para la actualización hace un ROLLBACK y nos da una alerta de que la operación fue cancelada y cuál es el id del material que no cuenta con stock suficiente. El primer paso es hacer un UPDATE en la tabla guitarras y le decimos que el campo stock es igual al stock actual sumado a la cantidad ingresada al llamar al SP donde el id de la guitarra es igual al id ingresado al llamar al SP (esto debe ser antes de abrir el cursor ya que sino entra en el loop). Ahora abrimos el cursor, declaramos al nombre bucle como LOOP, y hacemos un FETCH del cursor a las variables v_idmat y v_cantmat, gracias al loop hará el recorrido e ira guardando los valores que necesitamos. A continuación abrimos un IF y le decimos que si la variable v_finbucle es igual a 1 deje el loop, esto es gracias a que anteriormente hicimos el CONTINUE HANDLER para que cuando ya no encontrara resultados que coincidan ponga esta variable en 1. Por ultimo hacemos un UPDATE en la tabla materiales y le decimos que el stock va a ser igual al stock actual menos la cantidad ingresada cuando llamamos al SP, osea la cantidad de guitarras, por la variable v_cantmat que es la que va recorriendo la cantidad de materiales utilizados según el id de guitarra de la tabla materiales_guitarras donde el id del material coincida con la variable v_idmat que es la que va recorriendo los id de materiales según el id de guitarra en la tabla materiales_guitarras, todo esto gracias al loop. Finalizamos dejando el bucle y cerrando el cursor. Es una forma rápida para hacer varias actualizaciones en un solo paso, cuando terminamos el proceso de fabricación y vamos a ingresar más stock de un modelo también actualizamos el stock de los materiales que se utilizaron para todo.

- Stored procedure #5 “sp_actcosto_mat”: Sirve para actualizar el costo de un material (ya sea que haya bajado o subido su valor), y es de gran utilidad ya según el id del material selecciona todas las guitarras que utilicen el mismo y actualiza su precio de venta en base al nuevo costo del material. Los parámetros que va a recibir cuando lo llamamos son el id del material que queremos actualizar y su costo. Declaramos una variable llamada v_idguit que nos va a ayudar a obtener todos los id de guitarras que usen ese material gracias a un cursor y otra variable llamada v_finbucle que nos ayudara a detener el loop del cursor. Declaramos el cursor llamado cursorguit para que seleccione todos los id de guitarra de la tabla materiales_guitarras donde el id del material coincida con el id ingresado al llamar al SP, luego declaramos un CONTINUE HANDLER para NOT FOUND (no encuentras más resultados) y seteamos la variable v_finbucle con valor 1 para que detenga el loop. Primero hacemos un UPDATE en la tabla materiales y actualizamos el costo con el valor ingresado al llamar al SP donde el id de material coincida con el id ingresado al llamar al SP (es importante que esto sea primero ya que sino queda dentro del loop). Ahora si abrimos el cursor y declaramos al bucle como LOOP, luego hacemos un FETCH del cursor a la variable v_idguit, la cual va actualizando su valor con todos los id de guitarra que correspondan al material, ponemos un condicional IF para que cuando la
  variable v_finbucle sea igual a 1 haga un LEAVE del bucle, esto gracias al CONTINUE HANDLER declarado anteriormente. Como últimos pasos hacemos un UPDATE en la tabla guitarras y le decimos que actualice el precio (aquí utilizamos la función 'fx_calc_cst_guit' la cual nos va a ayudar a calcular el nuevo costo de la guitarra con el costo del material actualizado y como parámetro le vamos a pasar la variable v_idguit que es la que va recorriendo todos los id de las guitarras que hay que actualizar y lo multiplicamos por 1.7 para ya obtener nuestro precio final de venta) donde el id de la guitarra sea igual a la variable v_idguit que como ya dijimos es la que recorre todos los id de guitarras correspondientes. Para finalizar hacemos un END LOOP del bucle y cerramos el cursor.

- Stored Procedure Extra #1 “sp_ingresar_materiales_guitarras”: Esta creado pensando en la automatización del sistema, sirve para ingresar el detalle de materiales que lleva una guitarra. Al llamarlo ingresamos como parámetros el id del material, el id de la guitarra que utiliza ese material y la cantidad. Primero se declara una

variable para obtener el precio de venta de la guitarra por unidad llamada v_precio_venta, que luego la usaremos para actualizar la columna precio de la tabla guitarras y declaramos otra variable llamada v_mat_rest para guardar la cantidad de material que se utilizo en base al stock ingresado de la guitarra que luego usaremos para actualizar la columna stock de la tabla materiales. Declaramos un EXIT HANDLER para el error 1264 (out of range value), ya que el campo stock de la tabla materiales es UNSIGNED, osea que no puede ser negativo, por lo tanto si no disponemos stock del material hace un ROLLBACK dándonos un mensaje de alerta de que la operación ha sido cancelada. Hacemos un INSERT a la tabla materiales_guitarras con los datos ingresados al llamar al SP. Le damos valor a la variable v_precio_venta haciendo un SELECT del costo del material de la tabla materiales multiplicándolo por 1.7 donde el id del material sea igual al id_material ingresado al llamar al SP. Hacemos un UPDATE en la tabla guitarras indicando que el precio va a ser igual al precio que ya tiene sumado a la variable v_precia_venta donde el id de la guitarra sea igual al id_guitarra ingresado al llamar al SP. Le damos valor a la variable v_mat_rest haciendo un SELECT del stock de guitarras ingresado en la tabla guitarras por la cantidad de material utilizada que ingresamos al llamar al SP donde el id de la guitarra sea igual al id_guitarra ingresado al llamar al SP. En el último paso con este valor hacemos un UPDATE en la tabla materiales diciéndole que el stock va a ser igual al stock actual menos la variable v_mat_rest donde el id del material sea igual al id_material ingresado al llamar al SP. Es muy beneficioso para el sistema ya que cada vez que ingresamos un material a la guitarra va a ir sumando los costos de los materiales ya multiplicados por el valor total a marcar para la venta de productos y cuando finalicemos de cargar los materiales tendremos el valor de venta de la guitarra, y a su vez nos va a descontar los materiales utilizados para la fabricación dándonos un mensaje de alerta en el caso de que haya una inconsistencia entre el stock de guitarras ingresado y la cantidad de material que hay que utilizar.

- Stored Procedure Extra #2 “sp_ingresar_detalle_pedido”: Esta creado pensando en la automatización del sistema, sirve para ingresar el detalle de un pedido. Al llamarlo ingresamos como parámetros el id del pedido al cual vamos a ingresar el detalle, el id de la guitarra que se está pidiendo y la cantidad. Se declaran dos variables, una para obtener el precio de la guitarra por unidad llamada v_precio_guit y otra para obtener el importe total multiplicando el precio de la guitarra por la cantidad llamada v_importe_total. Declaramos un EXIT HANDLER para el error 1690 (BIGINT UNSIGNED) ya que la columna stock de la tabla guitarras no puede ser negativa, por lo tanto si no tenemos stock suficiente para el pedido nos dará un mensaje de alerta con el id de guitarra que no cuenta con stock, es muy ventajoso ya que podemos ingresar varios detalles de un pedido en una sola transacción y en el caso de que uno de los detalles no tuviera stock nos va a dar una alerta pero el resto va a quedar guardado, de esta forma tenemos la opción de ingresar otro modelo y cantidad, finalizarlo o inclusive hacer un ROLLBACK para borrar todo. Primero hacemos un UPDATE en la tabla guitarras diciéndole que el stock va a ser igual al stock actual menos la cantidad ingresada al llamar al SP donde el id de la guitarra sea igual al id_guitarra ingresado al llamar al SP, es el primer paso ya que si no contamos con stock el EXIT HANDLER anulara el resto del procedimiento. Como segundo paso hacemos un SELECT para obtener el precio de la guitarra de la tabla guitarras donde el id de la misma sea igual al ingresado al SP como parámetro y se lo damos como valor a la variable v_precio_guit, luego de eso hacemos un INSERT a la tabla pedidos_detalle con los datos ingresados al SP y el valor de v_precio_guit. Una vez que ya ingresamos el registro del detalle procedemos a darle valor a la variable v_importe_total haciendo un SELECT de la variable v_precio_guit por la cantidad ingresada en el SP obteniendo el precio total de este registro para finalizar haciendo un UPDATE en la tabla pedidos indicando que el campo precio_total va a tener el valor de este mismo campo sumado a la variable v_importe_total donde el id del pedido sea igual al id_pedido ingresado en el llamado a la SP. De esta forma cada vez que ingresamos un pedido al ir ingresando el detalle automáticamente se irán sumando los valores y cuando finalicemos tendremos el total en nuestra tabla pedidos, a su vez se descontara la cantidad pedida de nuestro stock de guitarras y se verifica que la misma cuente con stock suficiente.

LISTADO DE TRIGGERS

- Trigger #1 “ingreso_nuevo_pedido”: Guarda información de fecha y usuario que realizo el movimiento y el id y cliente del pedido sobre el cual se realizó cuando hacemos un INSERT en la tabla pedidos. Utilizamos las variables @nom y @ape para guardar el nombre y el apellido de la tabla clientes, luego lo concatenamos en la variable @dat que vamos a utilizar para poder insertar el nombre y apellido del cliente dentro del campo cliente de la tabla de movimientos, también utilizamos los métodos CURDATE, CURTIME y USER para obtener los datos de fecha, horario y usuario que realizó el movimiento. Es muy útil a futuro ya que podemos tener registros de quien fue el que ingreso un pedido en el caso de que se haya ingresado mal algún dato.

- Trigger #2 “borrar_pedido”: Guarda información de fecha y usuario que realizo el movimiento y el id y cliente del pedido sobre el cual se realizó cuando hacemos un DELETE en la tabla pedidos. Utilizamos las variables @nom y @ape para guardar el nombre y el apellido de la tabla clientes, luego lo concatenamos en la variable @dat que vamos a utilizar para poder insertar el nombre y apellido del cliente dentro del campo cliente de la tabla de movimientos, también utilizamos los métodos CURDATE, CURTIME y USER para obtener los datos de fecha, horario y usuario que realizó el movimiento. Es muy útil a futuro ya que podemos tener registros de quien fue el que elimino un pedido si se eliminó mal o cual había sido su valor.

- Trigger #3 “actualizar_costo_material”: Guarda información de fecha y usuario que realizo el movimiento y el id, tipo, costo viejo, costo nuevo y diferencia de costo en % del material sobre el cual se realizó cuando hacemos un UPDATE en la tabla materiales sobre el campo costo. Gracias al condicional IF donde ponemos que el OLD(viejo) costo tiene que ser diferente del NEW(nuevo) costo este trigger solamente se activara cuando haya actualizaciones de costo, también utilizamos los métodos CURDATE, CURTIME y USER para fecha, hora y usuario del movimiento y el método ROUND para redondear el resultado cuando hacemos el cálculo de la diferencia en porcentaje. Tiene 2 utilidades, la primera es poder saber quién realizo el movimiento en el caso de que haya habido algún error a la hora de modificar un precio, la segunda es poder saber en qué porcentaje aumento o disminuyo el costo del material.

- Trigger #4 “ingresar_material”: Guarda información de fecha y usuario que realizo el movimiento y el id, tipo y costo del material sobre el cual se realizó cuando hacemos un INSERT en la tabla materiales. Usamos los métodos CURDATE, CURTIME y USER para traer la fecha, hora y usuario que realizo el movimiento. Su beneficio es poder saber quién hizo el ingreso de un nuevo material al sistema en el caso de que se haya ingresado algún dato mal o poder saber en qué fecha ingreso por última vez un material para calcular cuánto hubo de gasto.