****

**FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN**

**DISEÑO DE SOFTWARE**

**POLIVENTAS**

**DANNY DE LA A YAGUAL**

**AXEL GODOY SAAVEDRA**

**FERNANDO MORENO MACIAS**

**JOS REYES RUIZ**

**Paralelo: 1**

**Profesor: David Jurado**

**TÉRMINO II 2018 – 2019**

Tabla de contenido

[Tabla de contenido 2](#_Toc535759657)

[Breve Introducción 3](#_Toc535759658)

[Patrones de diseño utilizados 3](#_Toc535759659)

[Reportes 4](#_Toc535759660)

# Breve Introducción

El presente reporte tiene como finalidad presentar puntos relevantes sobre el desarrollo del diseño de una plataforma para una empresa cuyo principal objetivo es unificar en un solo lugar a diferentes estudiantes emprendedores que buscan vender sus productos o servicios además que exista la posibilidad de receptar el pago a través de la plataforma utilizando conceptos básicos del diseño de software así como herramientas básicas con el fin de conocer el funcionamiento de la plataforma para implementarla en un futuro.

# Patrones de diseño utilizados

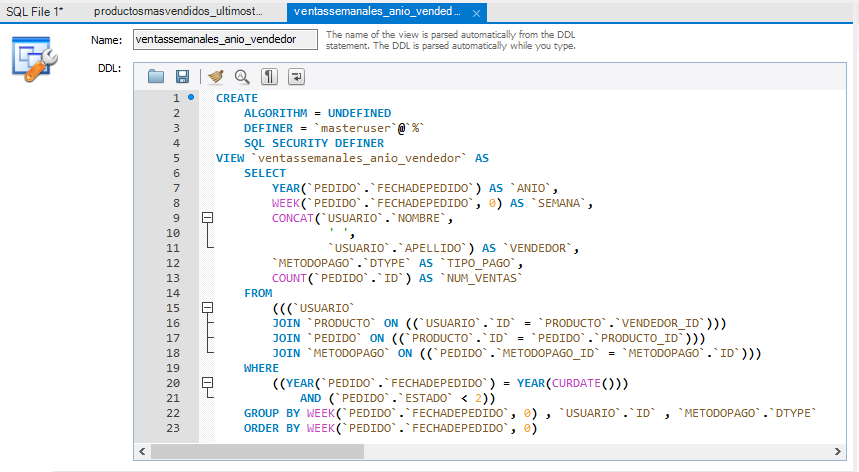
* Observer: Para implementar las notificaciones a los vendedores de que uno de sus productos ha sido comprado, las clases util::NotificadorEmail y util::NotificadorCelular implementan una variante de Observer. Estas clases son los observadores, y cuando un pedido es creado, notifica a uno o ambos de estos observadores, dependiendo de si el vendedor ha especificado o no su celular. Se diferencia del Observer normal porque los suscriptores (en este caso, los notificadores) no se registran con los productores (los Pedidos). Este paso es innecesario, porque los productores/Pedidos “empujan” los cambios directamente a la única instancia de los notificadores/suscriptores.
* Singleton:
  + Las clases util::NotificadorEmail y util::NotificadorCelular implementan Singleton, de modo que sólo hay una instancia de cada una, encargada de enviar correos electrónicos y mensajes SMS, respectivamente, sobre diversos temas de ser necesario. Esta decisión se debe a que estos notificadores no requieren que cada cliente tenga su propia copia para ser notificado, basta que exista una copia única.
  + La clase util::ConexionSQL implementa Singleton, porque en la aplicación sólo tiene que haber una conexión a la base de datos a la vez.
* Strategy: Las clases Models::MetodoPago y sus clases hijas, Models::PagoEfectivo y Models::PagoMonedero implementan Strategy, ya que la situación en la que se usan (ambas clases tiene que hacer la misma función, pero cada una la realiza de forma diferente) es el ejemplo típico del uso de Strategy. Esto simplifica la adición de nuevos métodos de pago: simplemente se crea una nueva clase que extienda Models::MetodoPago, con los datos que sean necesarios para confirmar el pago (por ejemplo, si fuera una tarjeta de crédito, el número de tarjeta, el nombre del dueño y el código de seguridad/CVV).

En nuestro Modelo no encaja ninguno de los patrones de diseño estructurales, por lo tanto, no se aplicó ninguno de ellos al sistema.

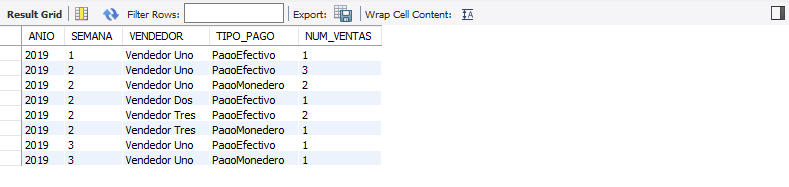
# Reportes

1. La cantidad de ventas semanales del año en curso, por tipo de pago y vendedor.

Query para crear vista.

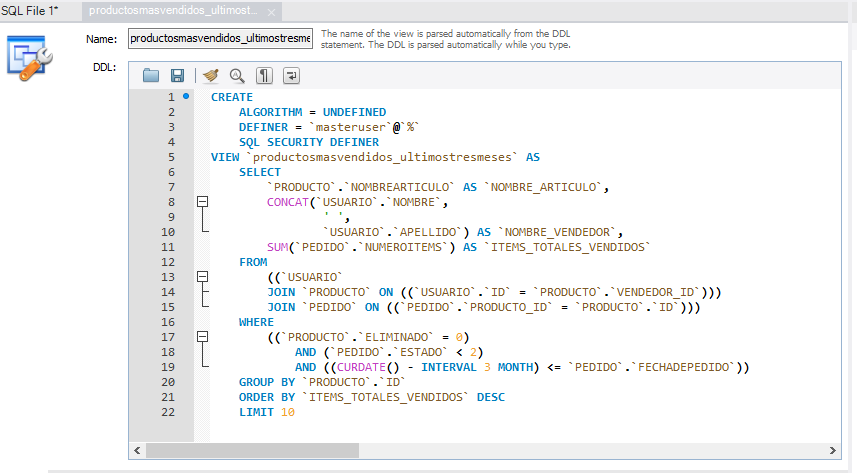


Resultado



1. Los 10 artículos más vendidos en los últimos 3 meses.

Query para crear vista.



Resultado

