**Speech - Parte de Franco Kaner Loy (Ajustado a ~6:25 min)**

**(Transición después de Bautista. Franco toma la palabra)**

**[Franco]:** "¡Muchas gracias, Bautista! Buenas tardes a todos. Soy Franco Kaner Loy, y continuaré con la presentación de nuestro sistema de gestión para el Taller Mecánico, mostrando cómo otras **vistas**, un **procedimiento almacenado** y un **trigger** específicos nos ayudan en la operación diaria."

**(Transición visual: Pantalla de SQL Server Management Studio (SSMS). Abre la pestaña de Vistas\_Taller\_Mecanico.sql.)**

**Vistas (Aprox. 2:45 minutos)**

**[Franco]:** "Siguiendo con el uso de las vistas, hemos implementado otras que son clave para el seguimiento de la productividad y la gestión de insumos."

**(Muestra el código de vw\_OrdenesFinalizadas en SSMS)**

**[Franco]:** "Mi primera vista es vw\_OrdenesFinalizadas. Esta vista es esencial para la administración, ya que nos permite **visualizar rápidamente todas las órdenes de trabajo que han sido completadas y entregadas al cliente**.

"**[Apuntando al código en pantalla, resalta el WHERE EO.Descripcion = 'Entregado']:** Como pueden ver, la lógica es sencilla pero efectiva: seleccionamos la información relevante de la Orden\_Trabajo, Cliente, Vehiculo y Estado\_Orden, y filtramos específicamente aquellas órdenes cuyo Estado es 'Entregado'."

**(Ejecuta SELECT \* FROM vw\_OrdenesFinalizadas; en SSMS y muestra los resultados brevemente)**

**[Franco]:** "Al ejecutarla, obtenemos una lista limpia de todas las órdenes que ya salieron del taller, con su número de orden, cliente, vehículo, fecha de ingreso, estado y el diagnóstico. Es un informe rápido para el cierre de mes o para verificar el flujo de trabajo."

**(Muestra el código de vw\_RepuestosPorOrden en SSMS)**

**[Franco]:** "Luego, tenemos la vista vw\_RepuestosPorOrden. Esta es crucial para el control de costos y el seguimiento de lo que se utiliza en cada trabajo. Nos **detalla todos los repuestos consumidos en cada orden de trabajo**."

"**[Apuntando al código, resalta las uniones con Orden\_Repuestos y Repuestos y la inclusión de Cantidad, PrecioUnitario y Cantidad \* PrecioUnitario]:** Aquí, unimos las tablas Orden\_Trabajo, Cliente, Repuestos y Orden\_Repuestos. Lo interesante es que no solo mostramos el nombre del repuesto, sino también la Cantidad utilizada, el PrecioUnitario y calculamos el subtotal de cada repuesto dentro de la orden (Cantidad \* PrecioUnitario), dándonos un desglose completo del costo de los materiales."

**(Ejecuta SELECT \* FROM vw\_RepuestosPorOrden; en SSMS y muestra los resultados)**

**[Franco]:** "El resultado nos permite ver, por ejemplo, que en la Orden 1 se usaron 2 unidades de 'Filtro de Aceite' a $2500 cada uno, totalizando $5000 por ese ítem. Esto es extremadamente útil para la elaboración de presupuestos detallados y para el seguimiento del consumo de materiales."

**Procedimiento Almacenado (Aprox. 1:30 minutos)**

**[Franco]:** "Pasemos ahora a un **Procedimiento Almacenado** que automatiza una acción muy importante en el día a día del taller: la asignación de personal a los trabajos."

**(Muestra el código de sp\_AsignarEmpleadoAOrden en SSMS)**

**[Franco]:** "Mi procedimiento almacenado es sp\_AsignarEmpleadoAOrden. Este procedimiento es fundamental para **gestionar la asignación de empleados a una orden de trabajo específica**, asegurando que solo se asignen empleados válidos y que no haya duplicidades."

"**[Apuntando al IF NOT EXISTS para empleado activo y el IF EXISTS para ya asignado, y el BEGIN TRY...CATCH]:** Aquí, la robustez es clave. Antes de insertar el registro en Orden\_Empleado, validamos dos cosas: primero, que el @idEmpleado que se intenta asignar exista y esté Activo. Si no, se lanza un mensaje de error. Segundo, que ese idEmpleado no esté ya asignado a esa idOrden para evitar duplicados. Además, incorporamos un bloque TRY...CATCH para un manejo de errores más elegante."

**(Prepara la demostración en SSMS):**

* **Antes:** SELECT idOrden, idEmpleado FROM Orden\_Empleado WHERE idOrden = 1 AND idEmpleado = 3  
  GO
* **Demostración de éxito:** EXEC sp\_AsignarEmpleadoAOrden @idOrden = 3, @idEmpleado = 1  
  GO
* **Después:** SELECT idOrden, idEmpleado FROM Orden\_Empleado WHERE idOrden = 1 AND idEmpleado = 3  
  GO
* **Caso de error:** EXEC sp\_AsignarEmpleadoAOrden @idOrden = 6, @idEmpleado = 1  
  GO (CASO DE ERROR)

**[Franco]:** "Para demostrarlo, primero confirmamos que el empleado 'Carlos Pereyra' con IdEmpleado 3 está activo. **[Muestra el SELECT del empleado activo]**. Ahora, vamos a asignarlo a la Orden 1. **[Ejecuta EXEC sp\_AsignarEmpleadoAOrden @idOrden = 1, @idEmpleado = 3;]**. Si todo es correcto, se inserta la asignación. **[Muestra el SELECT que evidencia la asignación]**. Podríamos intentar asignarlo de nuevo o asignar un empleado inactivo y verán cómo el procedimiento nos alerta con mensajes de error, manteniendo la consistencia de los datos."

**Trigger (Aprox. 1:30 minutos)**

**[Franco]:** "Para finalizar mi parte, les mostraré un **Trigger** que automatiza un proceso crucial en la vida de una orden: su cambio de estado al ser facturada."

**(Muestra el código de tr\_CambiarEstadoOrden en SSMS)**

**[Franco]:** "Mi trigger es tr\_CambiarEstadoOrden. Este se dispara **automáticamente después de insertar un nuevo registro en la tabla Facturas**. Su propósito es sincronizar el estado de la orden de trabajo con el proceso de facturación."

"**[Apuntando a AFTER INSERT ON Facturas y SET idEstado = (SELECT idEstado FROM Estado\_Orden WHERE Descripcion = 'Entregado')]:** Cuando una nueva factura es creada y registrada en la base de datos, este trigger detecta esa inserción. Inmediatamente, busca la Orden\_Trabajo asociada a esa factura y **actualiza su estado a 'Entregado'**. Esto significa que una orden solo pasa a 'Entregado' cuando ha sido efectivamente facturada, garantizando la integridad del flujo de trabajo."

**(Prepara la demostración en SSMS):**

* **Antes:** SELECT OT.idOrden, EO.Descripcion FROM Orden\_Trabajo OT JOIN Estado\_Orden EO ON OT.idEstado = EO.idEstado WHERE OT.idOrden = 2  
  GO  
   (Elige una orden que no esté 'Entregado' ni 'Cancelado', por ejemplo, 'En Proceso').
* **Acción:** INSERT INTO Facturas (Fecha, Total, idOrden) VALUES (GETDATE(), 10000.00, 2)  
  GO  
   (Asegúrate que la Orden 2 exista).
* **Después:** SELECT OT.idOrden, EO.Descripcion FROM Orden\_Trabajo OT JOIN Estado\_Orden EO ON OT.idEstado = EO.idEstado WHERE OT.idOrden = 2  
  GO

**[Franco]:** "Para ilustrarlo, tomemos la Orden 2, que actualmente se encuentra en estado **[Muestra el SELECT 'antes' - ej. 'En Proceso']**. Ahora, vamos a generar una factura para esta orden. **[Ejecuta el INSERT en Facturas]**. Una vez que la factura se registra, el trigger se activa."

"Si volvemos a consultar el estado de la Orden 2, **[Muestra el SELECT 'después' - ej. 'Entregado']**, verán que su estado ha cambiado automáticamente a 'Entregado'. Esto automatiza la finalización del ciclo de vida de la orden una vez que la transacción económica se ha completado."

**[Franco]:** "Con esto, concluyo mi demostración. A continuación, mis otros compañeros les mostrará los últimos componentes de nuestra base de datos. ¡Gracias!"